p. 46

Two Papers on Feed-Forward Networks

WRAY L. BUNTINE

wray@ptolemy.arc.nasa.gov
RIACS & AI Research Branch, Mail Stop 244–17
NASA Ames Research Center
Moffett Field, CA 94035, USA

ANDREAS S. WEIGEND

andreas@psych.stanford.edu
Jordan Hall (Building 420)
Stanford University, CA 94305–2130, USA

NASA Ames Research Center

Artificial Intelligence Research Branch

Technical Report FIA–91–22

July 5, 1991

REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Dates attached	

4. TITLE AND SUBTITLE

Titles/Authors - Attached

5. FUNDING NUMBERS

6. AUTHOR(S)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Code FIA - Artificial Intelligence Research Branch
Information Sciences Division

8. PERFORMING ORGANIZATION REPORT NUMBER

Attached

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Nasa/Ames Research Center

Moffett Field, CA. 94035-1000

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Available for Public Distribution

Peter Friedland 5/14/92 BRANCH CHIEF

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Abstracts ATTACHED

14. SUBJECT TERMS			15. NUMBER OF PAGES
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Two Papers on Feed-Forward Networks

WRAY L. BUNTINE

wray@ptolemy.arc.nasa.gov
RIACS & AI Research Branch, Mail Stop 244–17
NASA Ames Research Center
Moffett Field, CA 94035, USA

ANDREAS S. WEIGEND

andreas@psych.stanford.edu
Jordan Hall (Building 420)
Stanford University, CA 94305–2130, USA

Abstract

Connectionist feed-forward networks, trained with back-propagation, can be used both for non-linear regression and for (discrete one-of-C) classification, depending on the form of training. This report contains two papers on feed-forward networks. The papers can be read independently. They are intended for the theoretically-aware practitioner or algorithm-designer, however, they also contain a review and comparison of several learning theories so provide a perspective for the theoretician. The first paper works through Bayesian methods to complement back-propagation in the training of feed-forward networks. The second paper addresses a problem raised by the first: how to efficiently calculate second derivatives on feed-forward networks.

Contents

Chapter 1

Bayesian Back-Propagation

Connectionist feed-forward networks, trained with back-propagation, can be used both for non-linear regression and for (discrete one-of-C) classification, depending on the form of training. This paper works through approximate Bayesian methods to both these problems. Methods are presented for various statistical components of back-propagation: choosing the appropriate cost function and regularizer (interpreted as a prior), eliminating extra weights, estimating the uncertainty of the remaining weights, predicting for new patterns ("out-of-sample"), estimating the uncertainty in the choice of this prediction ("error bars"), estimating the generalization error, comparing different network structures, and adjustments for missing values in the training patterns. These techniques refine and extend some popular heuristic techniques suggested in the literature, and in most cases require at most a small additional factor in computation during back-propagation, or computation once back-propagation has finished. The paper begins with a comparative discussion of Bayesian and related frameworks for the training problem.

1.1 Introduction

Back-propagation [RHW86] is a popular scheme for training feed-forward connectionist networks. Recent improvements have looked at tasks such as speeding up learning using more sophisticated gradient descent algorithms [BL88, EJM90], eliminating insignificant weights in order to find networks of an optimal size [LDS90, WHR90, WRH91], and various modifications to apply a network to a one-of-C classification task (prediction of a single discrete variable taking one of C mutually exclusive and exhaustive values) rather than a non-linear regression task (prediction of a single real variable) [Bri89, DL91].

There is much to be gained by looking at the intersection between probabilistic techniques and theory, and feed-forward connectionist networks. On the one hand, feed-forward networks offer a broad computational framework for implementing a wide variety of probabilistic functions, and on the other hand, probability theory offer techniques for refining back-propagation schemes. We consider both these issues in this paper.

Since Bayesian probabilistic methods for learning tasks sometimes prove superior to other approaches on smaller samples [Bun90, Ber85, Bun91, OH91], we here frame the

probabilistic component of back-propagation in a Bayesian context. Back-propagation algorithms are known to work well in a variety of noisy and uncertain domains using a variety of different network structures and activation functions, so we propose Bayesian modifications and extensions to existing methods. Of course, in adopting a Bayesian justification for the methods presented, we are not claiming any neurological validity for our methods. We view feed-forward networks as a vehicle for massive parallelism in classification, regression and learning. This occurs because feed-forward networks can be constructed from simple component parts but still are able to represent a broad range of functions.

The beauty of the Bayesian approach arises from the fact that the entire method is derived as an approximation to applying the single simple formula

$$\textit{posterior} \propto \textit{prior} \cdot \textit{likelihood}$$

to the training problem, together with some principles for reasoning about the prior knowledge available.

Section 1.2 gives some theoretical background and motivation for using Bayesian methods, and discusses some alternative approaches. Section 1.3 details the notation and form of networks considered here. Section 1.4 outlines some network forms that correspond to variations of standard probability functions. A good feed-forward network system should therefore subsume the tasks of several special purpose statistical systems, albeit at some computational cost. Section 1.5 presents a probabilistic analysis of the training of feed-forward neural networks. Section 1.6 uses these results to present Bayesian embellishments for the standard back-propagation algorithm: cost functions, and weight–evaluation measures. Several minimum encoding approaches [WF87, BC91, Ris87] are also explained in Section 1.6.3. Finally, Section 1.7 discusses some extensions to back-propagation involving weight–elimination strategies, prediction of variables and generalization error, and handling missing values.

1.2 On Bayesian methods

Bayesian methods described here are based on estimating posterior probabilities for different network weights. The posterior corresponds to a relative measure of belief in the many possible network weights, similar to the statistical mechanics idea of an ensemble of networks [LTS89]. Statistical mechanics theory has been developed in the context of training a perceptron in a noise-free environment to estimate the generalization error [OH91, SST91]. While this is an impressive demonstration of the theory, we are concerned with developing Bayesian methods for more general feed-forward networks trained in noisy and/or uncertain environments, as commonly found in practice.

Bayesian and statistical mechanics methods both share the view that since we can never determine the "correct" or "best" weights, we should carefully reason about the reasonable possibilities. In Bayesian practice, however, such as [Mac91] more care is required in the selection of the prior and the error model, etc. We present methods for that here. Bayesian methods carefully separate the components of the learning problem: the priors on networks and network weights which represent our expectations before any training and correspond to regularizers; the utility or loss function for the problem which represent the goal of learning and correspond to quantities such as minimum errors or least squares; and the likelihood function for the network, such as a "Gaussian error model" as used in regression, which gives a statistical model of the network and how the data is expected to have been generated

initially. Notice that in general there is no "correct" prior, error model or likelihood, since by definition these vary from problem to problem and it is a challenge to try and make a choice that seems appropriate in a given circumstance. Notions such as the "true" subjective probability function [Gol88] or the "correct" likelihood function derived from the additive energy function [LTS89] are therefore only partially consistent with our general approach, though in several cases produce equivalent results. See also [Hau91] for a discussion in the uniform convergence framework.

Bayesian methods should not be confused with recent results that back-propagation methods are Bayes optimal [HP90]. Bayes optimal refers to the property that back-propagation methods should produce a network approaching the greatest lower bound on error (or more generally, risk) as the training sample size approaches infinity. Bayesian methods described here share this property but also have a more powerful property: They are approximating a method that produces a classifier/regression that will on average have equal or lower error (risk) than a classifier/regression produced by *any* other method applied to the same training sample. Notice, this holds for the current training sample, not just the infinite one considered by Bayes optimality. Also, notice, that this powerful property rests on certain assumptions, which are discussed below. This requires careful use of techniques in practice.

These properties hold because Bayesian methods are "normative" or "rational" [Ber85, HHL86]: any other approach not approximating them should not perform as well on average. As an example, minimum encoding methods such as MML [WF87] and MDL [Ris87] are often viewed as approximate Bayesian methods. Accordingly, a more exact Bayesian approach convincingly outperforms these encoding methods on the task of learning tree classifiers [Bun90] made popular by Quinlan's ID3 algorithm [Qui86].

Uniform convergence methods [BH89, Hau91], the basis for computational learning theory, approximate Bayesian methods when the sample size is large [Bun91]. Uniform convergence methods require a sample size large enough so that the prior term found in Bayesian methods becomes insignificant (and so can be ignored). Bayesian methods for connectionist networks are therefore an important complement to these and can provide keener insight about learning from smaller samples to the algorithm designer.

It is important to bear in mind, however, that Bayesian methods are not a replacement for uniform convergence methods, as indeed, they are not a replacement for other techniques such as cross-validation or minimum encoding methods, etc. All methods have particular algorithmic, complexity, and statistical properties that make them more appropriate in one engineering context or another. Cross-validation, for instance, can be fairly easy to implement on top of an existing algorithm so the algorithm can be got up and running quickly (even though its subsequent performance may be quite slow).

The theory behind Bayesian methods rests on two critical assumptions:

(I) the choice of models (or hypotheses) being searched must contain the "true" model (in practice this means a fairly accurate approximation to the "true" model),

(II) and the choice of prior over these models should represent a reasonable initial preference for models in the search space.

Although there is considerable literature on how to choose a prior to minimize the assumptions implicit in the prior [Ber85], and in practice we often consider a range of priors for their suitability [Mac91, Bun90]. Bayesian methods then guarantee best average case performance given these two assumptions and a third assumption:

(III) that the approximations made in implementation are sufficiently accurate.

Poor modeling will lead to strong but inappropriate assumptions. For example, a single layer perceptron network cannot capture certain higher-order functions, so is inappropriate for such tasks. So Bayesians say "A good Bayesian will usually out-perform a non-Bayesian but a bad Bayesian can do far worse" [Goo83]. In contrast, uniform convergence methods make no assumptions (the model space does not have to contain the "true" model and no prior is needed) but can only guarantee good performance in the worst case, so they sacrifice the guarantee of average case optimality. Notice that for larger samples, the worst case and the average case converge but for smaller samples the worst and average case can be far different [Bun91].

Non-Bayesians often say that "Bayesian methods require the assumption of a prior on networks which may not be known in practice". In fact, every algorithm or method is making implicit assumptions about a prior that can be explicitly calculated by running the algorithm on many different data sets and seeing the networks that result [Bun90]. In other words, some kind of prior assumption is unavoidable in practice, except where sample sizes are sufficiently large so that prior assumptions become submerged in the wealth of information from the sample. So rather than ignoring the unavoidable, Bayesian methods offer principles for developing a prior and evaluating a prior in a practical application. The intent of Bayesian methods then is to make all assumptions underlying a method explicit and then reconstructing the most rational method based on those assumptions. Techniques for making the Bayesian assumptions more robust also exist, and are referred to as robust statistics [Ber85], however we do not consider them here.

1.3 Multi-Layer networks

The networks we consider consist of a directed acyclic graph, giving the network structure, together with the activation functions at each unit or node, which relate inputs to activation output. A directed acyclic graph is one containing no directed cycles, so the function computed by the network is not computed using any fixed point equations. The network deals with real numbers internally, although the inputs may be discrete, represented as, say, reals 0 and 1.

Denote the input variables to the network as a vector of values x and denote the response variable which the network is intended to predict as y. In the case of regression, the network output corresponds to the predicted regression for y given the input variables. This corresponds to the expected value or mean of the real valued variable y conditioned on the values of the input variables x. In the case of one-of-C classification, the network outputs a conditional probability distribution over C possible (mutually exclusive and exhaustive) values for the discrete variable y, conditioned on the values of the input variables. The output comes from n nodes and corresponds to a vector of real values summing to 1. The i-th value is the estimated conditional probability that the output variable should have the i-th discrete value.

Connections (given as directed arcs on a graph) correspond to the flow of information between units or nodes. This can be in the forward direction during inference (classification or regression) and in the backward direction when calculating derivatives. So a connection from node n to m represents that node m receives node n's activation during the inference stage, often multiplied with a weight $w_{m,n}$. The nodes in the graph with no incoming connections correspond to input nodes for the neural net. Their value is given to the system from some external source.

Let the non-output nodes in the network be indexed $1, 2, \ldots$. In regression, the output node has activation o and in one-of-C classification the output nodes have activation $o_1, \ldots, o_C$. The activation of any non-output node n is denoted u_n which is a real number. The activation function for the node m is a function of the activations u_n for nodes n inputting to node m, and the function is parameterized by a vector of parameters w_m.

The exact details of the activation functions relating inputs to a nodes activation is not important for the analysis here, except that they are parameterized by network weights. A typical activation function for a node n with vector of weights w_n would be the sigmoid acting on the weighted sum of its inputs $u_1, \ldots, u_l$,

$$\begin{aligned} u(u_1, \ldots, u_l) &= \frac{1}{1 + e^{-a \sum_{i=1,\ldots,l} w_{n,i} u_i + b_n}} \\ &= \frac{1}{2}\left(1 + \tanh \frac{a}{2}\left(\sum_{i=1,\ldots,l} w_{n,i} u_i + b_n\right)\right) . \end{aligned}$$

The slope of the sigmoid, a, can be absorbed into weights and bias without loss of generality and is set to one. The bias b_n for the node is denoted $w_{n,0}$. Some other activation functions that have been used are independent Gaussians found in radial basis functions, logistic, exponential and various trigonometric functions [Cot90].

In the case of one-of-C classification, the output activations need to be non-negative and sum to one. To do this, the output activation functions may normalize a set of non-negative activations from nodes in the previous layer $u_1, \ldots, u_C$, using the function

$$o_i(u_1, \ldots, u_C) = \frac{u_i}{\sum_{i=1,\ldots,C} u_i} .$$

We call this function a normalizing activation function. A related activation function is the Softmax function of Bridle [Bri89] given by

$$o_i(u_1, \ldots, u_C) = \frac{e^{u_i}}{\sum_{i=1,\ldots,C} e^{u_i}} .$$

1.4 Probabilistic neural networks

Two key issues in network design are whether the network can sufficiently approximate the "true" function that needs to be computed, and whether the network structure and weights has an interpretation or intrinsic form that can be explained as the "knowledge" in the network rather than treating the network as a black box. This second issue is more important than one might think. If network weights have some clear interpretation then we can configure the network in coherent ways, an expert can view the knowledge represented by the network, and we can more readily assign meaningful priors or some other initialization for the network weights [TSN90]. Probabilistic interpretations of networks is also discussed in [Bri89].

In this section, we present network structures where these issues are well understood. One of the advantages of the general network approach, however, is that the one network package can be used to implement all these network types and more, just be replacing the network activation functions and their derivatives. While we call these probabilistic neural networks, they should not be confused with the probabilistic networks that are used in

artificial intelligence [Pea88]. The neural networks are probabilistic in the sense that they implement a well-understood probability function.

1.4.1 Logistic networks

Suppose all variables are discrete, the problem is classification, and the problem is distinguished by certain marginal probabilities of key features. For instance, for boolean variables $x_0, x_1, \ldots, x_5$, the key features might be x_0x_3, x_1x_4, $x_3\overline{x_4}$, etc., and the distinguishing characteristics of the problem are the marginal probabilities $Pr(x_0x_3)$, $Pr(x_1x_4)$, $Pr(x_3\overline{x_4})$, etc. All other marginal probabilities are assumed to to follow somehow from these. A standard method of inference is to find the distribution which assumes the least amount of information apart from the importance of the given marginals. This is termed the maximum entropy argument and the class of distributions so derived belong to the general exponential family [MN89], of which the Gaussian and logistic functions are a special case.

Suppose the key features are represented by boolean functions $c_1, \ldots, c_f$ on the input and output variables (x, y) that return 1 if the feature holds and 0 otherwise. Then the exponential distribution takes the form

$$Pr(x, y) = \exp\left(\alpha_0 + \sum_{i=1,\ldots,f} \alpha_i c_i(x, y)\right) \quad ,$$

where α_0 is a normalizing constant and α are some parameters that determine $Pr(c_i(x, y))$. Since we are interested in the conditional probability distribution $Pr(y \mid x)$, we take the conditional version of this to get

$$Pr(y \mid x) \;=\; \frac{\exp\left(\sum_{i=1,\ldots,f} \alpha_i c_i(x, y)\right)}{\sum_y \exp\left(\sum_{i=1,\ldots,f} \alpha_i c_i(x, y)\right)} \; . \tag{1.1}$$

For y boolean this can be further simplified to

$$Pr(y \mid x) \;=\; \frac{1}{1 + \exp\left(\sum_{i=1,\ldots,f} \alpha_i (c_i(x, 0) - c_i(x, 1))\right)} \; ,$$

which is the general case of multivariate logistic regression (or logit) [Ame85]. In the simplest case with boolean variables, the distinguishing features are yx_i, $y\overline{x_i}$, $\overline{y}x_i$ and $\overline{yx_i}$ for the input variables x_i. That is, the distribution is fully determined by the probabilities of $Pr(x_i \mid y)$. The distribution reduces to the form

$$Pr(y \mid x) \;=\; \frac{1}{1 + \exp\left(\beta_0 + \sum_{i=1,\ldots,a} \beta_i x_i\right)} \; ,$$

thus recovering the sigmoid function. More complex cases introduce higher-order terms into the sum such as x_1x_5, and $x_2\overline{x_3}x_5$, etc. By introducing enough higher-order terms, any conditional probability distribution on boolean variables can be represented. In practice, higher-order terms would have to be carefully chosen to represent those kinds of features expected in the data.

These functional forms can be implemented as networks in the following manner. For a boolean output variable y, one hidden layer computes the higher order terms from the

input variables and the output node takes the weighted sum of these through a sigmoid function. In the general case there would have to be a second hidden layer after the first to compute the exponentials in Equation (1.1) and the final output layer has normalizing activation functions. We refer to both these kinds of networks as logistic networks.

1.4.2 Cluster networks

Cluster networks perform one-of-C classification, so they output a probability vector for the discrete output variable y. In the case of discrete inputs, cluster networks can be viewed as an extension of logistic networks where the higher-order terms in the network are discovered by the network during training of the first layer of hidden nodes. The first layer of hidden nodes detect hidden mutually exclusive and exhaustive classes in the data of the form usually found using unsupervised learning or clustering techniques [Pan89]. For instance, if input variables are all boolean then each node in the first layer can correspond to the partial occurrence of a particular noisy conjunct. If input variables are continuous, then nodes in the first layer can correspond to radial basis nodes. The detected classes are, however, fed into the second hidden layer which then constructs an unnormalized probability vector for y, to be normalized by the final output nodes.

Cluster networks implement a probability distribution based on the assumption that there is a hidden variable making all other variables independent. We assume there is a hidden variable z with K mutually exclusive and exhaustive values numbered 1 to K. All other variables, the input variables $x_1, \ldots, x_A$ and the output variable y are independent given z, so the model for the joint probability of $z, x_1, \ldots, x_A, y$ is of the form

$$Pr(z, x_1, \ldots, x_A, y) \;=\; Pr(z) Pr(y \mid z) \prod_{i=1,\ldots A} Pr(x_i \mid z) \quad .$$

For K large enough and all variables discrete, this family of distributions can arbitrarily approximate any other. The hidden variable z in practice has an unknown value and we are interested in the conditional probability of y given x, so we manipulate this joint to produce

$$Pr(y \mid x) \;=\; \frac{\sum_{j=1,\ldots K} Pr(y \mid z=j) Pr(z=j) \prod_{i=1,\ldots A} Pr(x_i \mid z=j)}{\sum_{j=1,\ldots K} Pr(z=j) \prod_{i=1,\ldots A} Pr(x_i \mid z=j)} \quad .$$

Notice that if the value for x_i is missing, then the corresponding entry in the product $\prod_{i=1,\ldots A}$ can be dropped and the formula still holds. If a variable x_i is continuous, we can let $Pr(x_i \mid z=j)$ be a Gaussian distribution with mean $\mu_{i,j}$ and standard deviation $\sigma_{i,j}$. In practice, we might want to use a totally different distribution for x_i. The current choice is sufficient for presentation.

The following network implements this family of distributions. The network has two hidden layers with full interconnections between layers. The output nodes (numbered layer 3) have normalizing activation functions, and represent the normalized conditional probability vector for y given x. The second hidden layer has C nodes, one for each value of the output variable y, and has linear activation functions. These activations represent an unnormalized conditional probability vector for y given x. These give activation u_i for $i = 1, \ldots, C$ corresponding to

$$u_i \;\equiv\; \sum_{j=1,\ldots K} Pr(y \mid z=j) Pr(z=j) \prod_{i=1,\ldots A} Pr(x_i \mid z=j) \; .$$

The first hidden layer has K nodes, each one corresponding to a hidden class, with exponential activation functions. These activations represent an unnormalized conditional probability vector for z given x. These give activation u_{C+j} for $j = 1, \ldots, K$ corresponding to

$$u_{C+j} \equiv Pr(z = j) \prod_{i=1,\ldots A} Pr(x_i \mid z = j) .$$

The bottom layer is the input layer. The network form is represented in Figure 1.1.

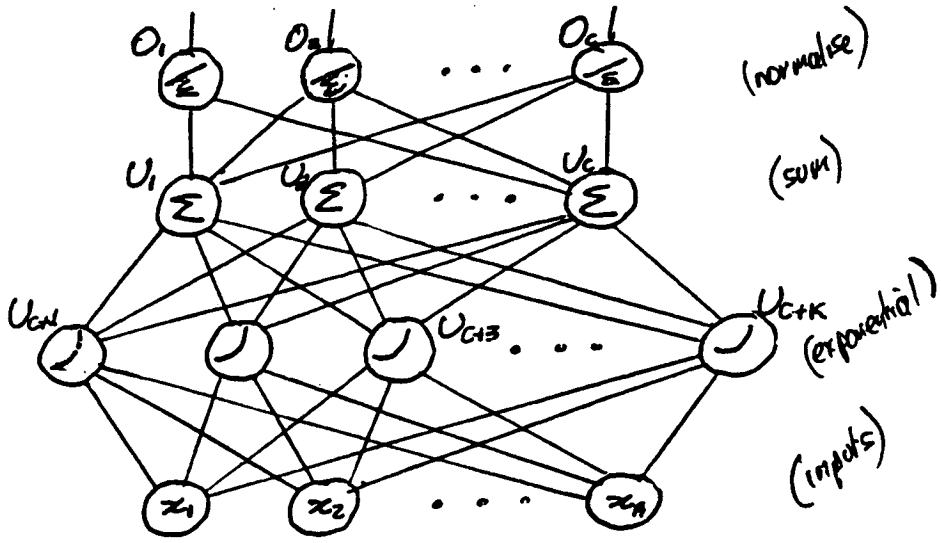

Figure 1.1: A cluster network

Activation functions are as follows:

$$o_i = \frac{u_i}{\sum_{j=1,\ldots,C} u_j} \qquad \text{for } i = 1, \ldots, C,$$

$$u_i = \sum_{j=1,\ldots,K} w_{i,C+j} u_{C+j} \qquad \text{for } i = 1, \ldots, C ,$$

$$u_{C+j} = \exp\left(w_{C+j,0} + \sum_{i \in discrete} \sum_{h=1,\ldots,d_i} w_{j,i,h} 1_{x_i = h} + \sum_{i \in continuous} \left(\frac{(x_i - \mu_{i,j})^2}{2\sigma_{i,j}^2} - \frac{1}{2} \log(2\pi\sigma_{i,j}^2) \right) \right) \qquad \text{for } j = 1, \ldots, K .$$

Notice the standard w notation for weights has been dropped in the case of $\mu_{i,j}$ and $\sigma_{i,j}$. The weights have the following interpretation:

$$\begin{aligned} w_{i,C+j} &\equiv Pr(y = i \mid z = j) , \\ w_{j,i,h} &\equiv \log Pr(x_i = h \mid z = j) , \\ w_{C+j,0} &\equiv \log Pr(z = j) . \end{aligned}$$

In addition, $\mu_{i,j}$ and $\sigma_{i,j}$ represent the mean and standard deviation of the Gaussian on x_i.

Notice also that the following constraints hold:

$$\sum_{i=1,\ldots C} w_{i,C+j} = 1 \quad , \quad \sum_{j=1,\ldots K} e^{w_{j+C,0}} = 1 \quad , \quad \sum_{h=1,\ldots d_i} e^{w_{j,i,h}} = 1 \quad ,$$

and $w_{i,C+j}$ are non-negative and $w_{C+j,0}$ are negative. These constraints can easily be enforced by restricting the changes made to these network weights during back-propagation. This can be done, for instance, using Lagrangian multipliers [LeC88], or simply by insuring that one weight is determined by all the others according to the constraint.

As an example of these network, suppose variables y and $x_1, \ldots, x_5$ are boolean. Then the conjunct $x_1 \wedge \overline{x_3} \wedge x_5$ is represented at the j-th node of the first layer by having $Pr(x_1 \mid z = j)$, $Pr(x_5 \mid z = j)$ and $Pr(\overline{x_3} \mid z = j)$ close to 1 and making $Pr(x_2 \mid z = j)$ and $Pr(\overline{x_4} \mid z = j)$ neither near 0 or 1, say about 0.5. The activation u_j will then be near zero if an input pattern fails to match the conjunct and closer to 0.25 if the conjunct is matched. Several such conjuncts can be represented, and so expressions in disjunctive normal form (a disjunction of conjunctive terms) are approximately represented.

1.4.3 Regression networks

A statistical model that has a simple network representation is linear regression [HT90], where the mean of the response variable is usually given by

$$y = \sum_{i=1,\ldots,K} w_i f_i(x) \, ,$$

for "basis functions" f_i and "parameters" or weights w_i. A corresponding network has K hidden nodes without weights which compute the functions $f_i(x)$, leading into linear output nodes that compute the linear regression. The regression is called linear because it is a linear function of the weights w_i, whereas the functions f_i can be arbitrary.

The various techniques such as smoothing, regularization, cross validation, etc., applied to these systems can then be cast in a network framework. MacKay presents a Bayesian model for this [Mac91]. The choice of basis functions depends on the behavior of y anticipated. Some choices are products of $\sin n_i x_i$ and $\cos n_i x_i$ for integer n_i when $x_i \in [-\pi, \pi]$; or combinations of Legendre polynomials or their integrals when $x_i \in [-1, 1]$ and y is expected to be in the form of a polynomial; or combinations of Hermite functions when $x_i \in [-\infty, \infty]$ and y is expected to be in the form of a product of $\exp(-\sum_{i=1}^{A} x_i^2/2)$ and some polynomial in x. Notice each of these choices may require rescaling the x_i's initially.

Of course in the more general case, we can use a semi-linear model

$$y = \sum_{i=1,\ldots,K} w_{0,i} f_i(x, w_{i,0}) \, ,$$

which again has a straight forward network interpretation.

1.5 Probabilistic Analysis

This section covers each component of a Bayesian analysis in turn: interpreting networks as a probability function in Section 1.5.1; thereby calculating the likelihood of the training

sample for a given network and weights, in Section 1.5.2; considering priors for the weights in a network, in Section 1.5.3; and finally considering posteriors for the weights in a network, in Section 1.5.4.

The notation $\mathbf{E}_{A|B}(f(A,B))$ denotes the expected (mean) value of the function $f(A,B)$ when A is distributed according to the probability function or probability density function $Pr(A \mid B)$. If A is continuous, this is calculated with an integral

$$\mathbf{E}_{A|B}(f(A,B)) \equiv \int_A f(A,B)\, Pr(A \mid B)\, dA ,$$

and if A is discrete this is calculated as

$$\mathbf{E}_{A|B}(f(A,B)) \equiv \sum_A f(A,B)\, Pr(A \mid B) .$$

For mixed continuous and discrete variables, combinations of these formula apply. Similarly, the notation $\mathbf{v}_{A|B}(f(A,B))$ denotes the variance of $f(A,B)$, usually calculated as

$$\mathbf{v}_{A|B}(f(A,B)) \equiv \mathbf{E}_{A|B}\left(\left(f(A,B) - \mathbf{E}_{A|B}(f(A,B))\right)^2\right) .$$

1.5.1 The network likelihood function

To do Bayesian or likelihood analysis on feed-forward networks, we have to use the network output $o(x,w)$, a function of the inputs x and the weights w, to construct a likelihood function $l(y \mid x,w)$ for the observed pattern (x,y). This likelihood function gives the conditional probability distribution for the output variable y conditioned on the input variables x, given a particular network and weights. This likelihood function is the basis of all subsequent analysis.

In the case of classification, the likelihood function is a probability distribution representing the network's estimate of the "true" conditional distribution for y given x. The C network outputs therefore give a conditional probability vector. The i-th output $o_i(x,w)$ represents the conditional probability that the discrete output variable y takes its i-th value. So

$$l(y \mid x,w) = o_y(x,w) .$$

In the case of regression, the likelihood function is a probability density function (it integrates over the domain of y to 1) of the form $l(y \mid x,w,F)$ for the output variable y conditioned on x, the network weights and some other information F such as a standard deviation of error. The network output is usually defined to be the mean of the likelihood function given by

$$o(x,w) \equiv \int_y y\, l(y \mid x,w,F)\, dy .$$

The likelihood is usually defined in terms of some *error model* that is a function of y, the mean $o(x,w)$, and some other error parameters. An error model commonly used is the Gaussian distribution with mean $o(x,w)$ and standard deviation σ that is unknown (so has to be estimated along with the weights w). This means the true y is expected to vary about the mean $o(x,w)$ with standard deviation σ. It is more realistic that the standard deviation itself should be a function of x too. In this case, a second output node can be connected to the network to estimate the standard deviation; we denote its output by $o'(x,r)$, and

note that the weights r will have parameters in common with w. Vapnik [Vap82] has also suggested using the Laplace distribution as an error model when the experimental conditions may vary with maximal uncertainty but the standard deviation is still independent of x. This leads to the following choice of error models: with known/unknown standard deviation σ of

$$l_G(y \mid x, w, \sigma) \equiv \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - o(x,w))^2}{2\sigma^2}\right) ;$$

with x dependent standard deviation $\sigma'(x,r)$ of

$$l_{GD}(y \mid x, w, r) \equiv \frac{1}{\sqrt{2\pi}\sigma'(x,r)} \exp\left(-\frac{(y - o(x,w))^2}{2\sigma'(x,r)^2}\right) ;$$

and with experiment dependent standard deviation of

$$l_L(y \mid x, w, \Delta) \equiv \frac{1}{2\Delta} \exp\left(-\frac{|y - o(x,w)|}{\Delta}\right) .$$

1.5.2 The sample likelihood

Given a training sample, we need to look at the likelihood of the sample as a function of the network parameters w. This assumes the weights w and any other parameters are known and gives the likelihood that the sample seen would occur. Bayesian learning equations are presented in the next section that show how this leads to learning of the weights.

For a sample consisting of N examples given as a list of input vectors $\boldsymbol{x} \equiv x_1, \ldots, x_N$ and corresponding output values $\boldsymbol{y} \equiv y_1, \ldots, y_N$, the sample likelihood in the classification problem (assuming examples are independently and identically distributed) is given by

$$L(\boldsymbol{y} \mid \boldsymbol{x}, w) = \prod_{i=1,\ldots,N} o_{y_i}(x_i, w) .$$

The sample likelihood in the regression problem with Gaussian error and standard deviation σ is given by

$$L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) = \prod_{i=1,\ldots,N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - o(x_i,w))^2}{2\sigma^2}\right) = \frac{1}{(2\pi)^{\frac{N}{2}}\sigma^N} \exp\left(-\frac{Ns^2}{\sigma^2}\right) ,$$

where

$$s^2 \equiv \frac{1}{N} \sum_{i=1,\ldots,N} (y_i - o(x_i,w))^2$$

is the observed average squared error. The sample likelihood for the Laplacian error model is similar.

Maximum likelihood methods for network training use the negative logarithm of the likelihood as a "cost" function (to be minimumized) [BW87, EJM90].

1.5.3 Prior probability of the weights

In the classification case, prior probabilities over the network weights w give our *a priori* belief that $o_i(x', w)$ for $i = 1, \ldots, C$ is the "true" distribution for y given x. In the regression case, prior probabilities over the network weights w give our *a priori* belief that $o(x', w)$ is

the "true" mean of y given x. The choice of prior represents an assumption about what form w should take. By representing the prior as a probability distribution $Pr(w)$, this prior assumption is not expressed as hard constraints such as $w_{n,m} = 1$ or $0 < w_{n,m} < 0.5$, but rather as preferences among the many different possible values.

There is considerable literature written about priors (see [Ber85] and references therein), and a number of methods exist for obtaining priors that are considered "non-informative" about the class of theories being considered. We suggest a few priors here, however believe more experience and research is needed for designing priors on neural networks. In general there is no "correct" prior, however, it is important to use a prior that has reasonable properties. See [Bun90] where a number of different priors are discussed for trees and [Mac91], where several priors are tried and compared for a single network learning problem.

When dealing with probabilistic neural networks, as given in Section 1.4, we can often use the probabilistic interpretation of the network to guide us in the choice of prior. Some examples are given below.

Priors on cluster networks

For instance, the cluster networks of Section 1.4.2 have weights corresponding to Gaussian parameters or proportions. The parameters $w_{i,C+j}$ for $i = 1, \ldots, C$ are parameters for a multinomial distribution, so $\sum_{i=1,\ldots,C} w_{i,C+j} = 1$. A standard non-informative prior for this is [Ber85, Bun90]

$$Pr(w_{1,C+j}, \ldots, w_{C,C+j}) \propto \prod_{i=1,\ldots,C} w_{i,C+j}^{1/C-1} ,$$

where we have ignored the normalizing constant so the prior is "$\propto$" rather than "$=$". Similarly, suitable priors for the other parameters can be found by using standard non-informative priors for the class of parameters concerned. Notice transformed parameters $e^{w_{C+j,0}}$ are multinomials, so we use the same form of multinomial prior as before for these. However we need a prior on weights $w_{C+1,j}$, not their exponential. This requires a change of variables, and note that $w_{C+K,0}$ is a function of $w_{C+1,0}, \ldots, w_{C+K-1,0}$. Likewise for $e^{w_{C+j,i,h}}$.

$$\begin{aligned}
Pr(w_{C+1,0}, \ldots, w_{C+K-1,0}) &\propto \exp\left(1/K \sum_{j=1,\ldots,K} w_{C+j,0} - w_{C+K,0}\right) , \\
Pr(w_{C+j,i,1}, \ldots, w_{C+j,i,d_i-1}) &\propto \exp\left(1/d_i \sum_{h=1,\ldots,d_i} w_{C+j,i,h} - w_{C+j,i,d_i}\right) , \\
Pr(\mu_{i,j}, \sigma_{i,j}) &\propto \frac{1}{\sigma_{i,j}} .
\end{aligned}$$

Priors on regression networks

For regression networks, priors say how we expect the mean of y given x to change with x. A standard approach is to say we expect y to vary smoothly as the input variables $x_1, \ldots, x_A$ change. For instance, for linear regression networks, one can say the average second derivative of y w.r.t. x should not be too large. This section argues that a suitable

prior on the weights takes the form

$$Pr(w) \propto C(w)^{-K/2}$$

for a quadratic function $C(w)$ on the weights w determined by the basis functions used in the regression. This is equivalent to but simplifies the approach given in [Mac91] for corresponding priors.

With x defined over the multi-dimensional space X, the average magnitude of the second derivative can be expressed as a function $C(w)$ given by

$$C(w) = \int_{x \in X} \left\| \frac{\mathrm{d}^2 y}{\mathrm{d}x^2} \right\| \mathrm{d}x \ ,$$

where the norm $\|\cdot\|$ is an average measure of the size of the second derivative at a single point. We assume the space X is finite but relax this condition later on. One way to calculate the norm $\|\cdot\|$ at the point x is to calculate the average change in y about a ball in x-space of fixed radius from x (i.e. $x + \Delta x$ where $|\Delta x| = 1$) due to the second derivative. Using a second order expansion, this is given as:

$$\begin{aligned}
\left\| \frac{\mathrm{d}^2 y}{\mathrm{d}x^2} \right\| &= \int_{|\Delta x|=1} \left(\frac{1}{2} \sum_{i,j=1}^{A} \Delta x_i \Delta x_j \frac{\partial^2 y}{\partial x_i \partial x_j} \right)^2 \mathrm{d}\Delta x \Big/ \int_{|\Delta x|=1} \mathrm{d}\Delta x \\
&= \left(\frac{\mathrm{d}^2 y}{\mathrm{d}x_1^2} \right)^2 \qquad \text{for } A{=}1, \text{ and for large } A \\
&\approx \frac{1}{4A^2} \left(\sum_{i,j=1}^{A} \frac{\partial^2 y}{\partial x_i^2} \frac{\partial^2 y}{\partial x_j^2} + 2 \sum_{i,j=1}^{A} \left(\frac{\partial^2 y}{\partial x_i \partial x_j} \right)^2 \right) \ .
\end{aligned}$$

We use the approximation below since it is proportional to the exact calculation when $A = 1$. Substituting into $C(w)$ and using the regression model $y = \sum_{i=1,\ldots,K} w_i f_i(x)$, we then get

$$C(w) = \sum_{m,n=1}^{K} w_m w_n C_{m,n} \ ,$$

where

$$C_{m,n} = \frac{1}{4A^2} \sum_{i,j=1}^{A} \int_{x \in X} \left(\frac{\partial^2 f_m(x)}{\partial x_i^2} \frac{\partial^2 f_n(x)}{\partial x_j^2} + 2 \frac{\partial^2 f_m(x)}{\partial x_i \partial x_j} \frac{\partial^2 f_n(x)}{\partial x_i \partial x_j} \right) \mathrm{d}x \ .$$

We can then put a prior on w in the general form

$$Pr(w \mid \alpha) \propto \exp\left(-\frac{\alpha}{2} C(w) \right) \ ,$$

where α is a prior "hyper-parameter" to be determined. A large value of α means we have a strong bias towards smooth functions, and a small value means otherwise. We leave α to be an undetermined parameter because we cannot be sure *a prior* how a particular value of α will effect smoothness. MacKay shows how to determine α *a posterior*, however we show below how to marginalize it out.

This gives a prior in the form of a multivariate Gaussian

$$Pr(w \mid \alpha) \;=\; \frac{\alpha^{K/2}\det{}^{1/2} C}{(2\pi)^{K/2}} \exp\left(-\frac{\alpha}{2}C(w)\right) \quad ,$$

where $K = |w|$ is the dimensionality of the weight set w. Since we are not really sure what $\alpha \in [0, \infty)$ should be anyway, we can marginalize it out. Since α appears as a factor of $C(w)$, it is a scaling quantity so a suitable prior is [Ber85]

$$Pr(\alpha) \;\propto\; 1/\alpha \quad .$$

This gives

$$\begin{aligned} Pr(w) \;&=\; \int_0^\infty Pr(w \mid \alpha) Pr(\alpha) \mathrm{d}\alpha \\ &\propto\; \frac{\Gamma(K/2)\det{}^{1/2} C}{\pi^{K/2} C(w)^{K/2}} \;\propto\; C(w)^{-K/2} \quad . \end{aligned}$$

During gradient descent, the important contribution is

$$\frac{\partial - log Pr(w)}{\partial w_i} \;=\; K/2 \frac{\partial \log C(w)}{\partial w_i} \;=\; \frac{K}{2C(w)} \frac{\partial C(w)}{\partial w_i}$$

If we compare this with the derivative of the original $Pr(w|\alpha)$, then it follows that

$$\alpha \;\approx\; \frac{K}{C(\widehat{w})} \; ,$$

where $\widehat{w}$ is the weights actually used. Since $C(w)$ is the measure of smoothness (where smaller is smoother), this says that one should change weights to increase smoothness of the mean function y. Notice from the definition of $C(w)$ that the partial derivatives $\frac{\partial C(w)}{\partial w_i}$ are calculated when calculating $C(w)$. Likewise for second derivatives. We therefore get that all first and second derivatives of $-log Pr(w)$ w.r.t. w are calculated with the same cost it takes to calculate $C(w)$.

Values of $C_{m,n}$ for a given set of basis functions only have to calculated once, but usually lead to a complex matrix. In this case, a diagonal approximation to the matrix may be sufficient. For instance, when using basis functions in the form

$$f_n(x_1, \ldots, x_A) \;=\; \prod_{i \in C_n} \sin n_i x_i \prod_{i \notin I_n} \cos n_i x_i$$

for different n_i and index sets I_n, then the diagonal term is given by

$$C_{n,n} \;\propto\; \left(\sum_{i=1}^{A} n_i^2\right)^2 \; .$$

Entropy priors

Zellner's maximal data information prior for a probability function $Pr(z \mid \theta)$ on parameters θ [Zel90] takes the form

$$Pr(\theta) \;\propto\; \exp\left(-I_{z|\theta}(\theta)\right) \quad ,$$

where

$$I_{x|\theta}(\theta) \;=\; -\int Pr(z|\theta)\log Pr(z|\theta)\,\mathrm{d}z$$

is the usual entropy function giving the entropy for z distributed as $Pr(z|\theta)$, which is a function of θ.

When doing classification, we are constructing a network for the conditional distribution $Pr(y \mid x, w)$. Assuming a distribution for x is known, a prior using the above method can then be formed as follows.

$$\begin{aligned} Pr(w) &\propto \exp\left(-\left(I_x + \mathbf{E}_x\left(I_{y|x,w}(w)\right)\right)\right) \\ &\propto \exp\left(-\mathbf{E}_x\left(I_{y|x,w}(w)\right)\right) \quad , \\ &\approx \exp\left(-\frac{1}{N}\sum_{d=1}^{N} I_{y|x_d,w}(w)\right) \quad . \end{aligned}$$

This assumes the entropy of x, I_x is independent of w. Since the distribution for x is not actually known, the last line estimates the expectation $\mathbf{E}_x(\cdot)$ with the empirical average from the sample of input vectors $x \equiv x_1, \ldots, x_N$. Since we are forming a prior on weights (parameters) for a conditional distribution of y given x, which presumably is independent of the distribution for x, we can use the sample of points for x to estimate a distribution for x. For the classification case,

$$I_{y|x_d,w}(w) \;=\; -\sum_{i=1}^{C} o_i(x_d, w)\log o_i(x_d, w) \quad .$$

For the regression case with x dependent standard deviation of $o'(x, r)$,

$$I_{y|x_d,w,r}(w, r) \;=\; \log o'(x_d, r) + constant \quad .$$

Weight elimination and priors

One particular prior we have worked with assume the weights have a prior probability that corresponds to

$$-\log Pr(w) \;=\; \lambda C(w) + \text{constant} \;=\; \lambda \sum_{l,m} \frac{w_{l,m}^2/w_0^2}{1 + w_{l,m}^2/w_0^2} + \text{constant} \quad . \qquad (1.2)$$

The unspecified constant term is required to ensure $Pr(w)$ normalizes to 1, but its value is unimportant since we primarily deal with derivatives. This term was first proposed by Rumelhart in 1987. w_0 is the scale for the weights: For $|w_{l,m}| \gg w_0$, the cost of a weight approaches unity (times λ), allowing the interpretation of the complexity term as a counter of significantly sized weights. For $|w_{l,m}| \ll w_0$, the cost is close to zero.

Relevant weights are drawn from a uniform distribution (to allow for normalization of the probability, up to a certain maximum size). Weights that are merely the result of "noise" are drawn from a Gaussian-like distribution centered on zero; they are expected to be small. The corresponding bump around zero can be approximated by a Gaussian with variance $\sigma^2 = w_0^2/\lambda$. The larger λ is, the closer to zero a weight must be to have a reasonable probability of being a member of the "noise" distribution.

The learning rule is then to change the weights according to the gradient of the *entire* cost function, continuously doing justice to the trade-off between error and complexity. This differs from methods that consider a set of fixed models, estimate the parameters for each of them, and then compare between the models by considering the number of parameters. Taking a complexity term into account during learning required λ to be adjusted dynamically during learning. The details of this procedure are described in [WRH91].

Bayesian interpretation of weight elimination

To place this approach in a Bayesian framework, consider the case where we place a prior on λ, $Pr(\lambda)$ to determine the overall prior on w. We do this as follows. We say

$$Pr(w \mid \lambda) \propto e^{-\lambda C(w)} \quad ,$$

where $C(w)$ is the sum given in Equation (1.2) and notice that this distribution is improper because it cannot be normalized ($C(w)$ is bounded above in magnitude by the number of weights $|w|$). If we restrict the weights w to remain inside a sufficiently large finite region W in w-space of area $|W|$, then we get

$$\begin{aligned}
Pr(w \mid \lambda) &\approx \frac{1}{|W|} e^{-\lambda(C(w)-|w|)} \quad , \\
Pr(w) &\approx \frac{1}{|W|} \int_\lambda p(\lambda)\, e^{-\lambda(C(w)-|w|)} \mathrm{d}\lambda \quad , \\
\frac{\partial - \log Pr(w)}{\partial w_{n,m}} &= \mu_{\lambda|w} \frac{\partial C(w)}{\partial w_{n,m}} \\
\frac{\partial^2 - \log Pr(w)}{\partial w_{n,m} \partial w_{l,k}} &= \mu_{\lambda|w} \frac{\partial^2 C(w)}{\partial w_{n,m} \partial w_{l,k}} + \sigma^2_{\lambda|w} \frac{\partial C(w)}{\partial w_{n,m}} \frac{\partial C(w)}{\partial w_{l,k}} \quad ,
\end{aligned}$$

where $\mu_{\lambda|w}$ is the expected value of λ given weights w and $\sigma_{\lambda|w}$ is its variance. Notice these derivatives are essentially the same as the derivatives of the original form in Equation (1.2) except that λ is now set automatically. The terms are given by

$$\mu_{\lambda|w} = \frac{\int_\lambda \lambda p(\lambda)\, e^{-\lambda(C(w)-|w|)} \mathrm{d}\lambda}{\int_\lambda p(\lambda)\, e^{-\lambda(C(w)-|w|)} \mathrm{d}\lambda} \quad ,$$

and $\sigma_{\lambda|w}$ is defined similarly. Reasonable values appear to be of the form

$$\begin{aligned}
\mu_{\lambda|w} &= \frac{1}{C(w)} \\
\sigma_{\lambda|w} &= \mu_{\lambda|w} \quad ,
\end{aligned}$$

which means λ is expected to vary inversely with the number of small weights.

1.5.4 Posterior analysis

To do Bayesian analysis, we need to be able to determine relevant posteriors from priors and likelihoods. For the regression case with Gaussian error and unknown standard deviation σ this is as follows. The product rule for probabilities gives the following:

$$L(y \mid x, w, \sigma) Pr(w, \sigma) Pr(x \mid w, \sigma) = Pr(w, \sigma \mid x, y) Pr(y \mid x) Pr(x)$$

where L is the sample likelihood as before, $Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y})$ equals the posterior probability of the network weights w and the parameter to the error model σ. $Pr(\boldsymbol{x} \mid w, \sigma)$ is the probability of seeing the inputs $\boldsymbol{x}$ given w and σ. This can be assumed independent of w and σ since they just parameterize the model for $\boldsymbol{y}$ given $\boldsymbol{x}$, so $Pr(\boldsymbol{x} \mid w, \sigma) = Pr(\boldsymbol{x})$ which is the prior probability of seeing the set of inputs $\boldsymbol{x}$. $Pr(\boldsymbol{x})$ then cancels out both sides. While $Pr(\boldsymbol{y} \mid \boldsymbol{x})$ also has a particular meaning, for our purposes it is a normalizing constant because it is independent of w and σ. Re-expressing this, and assuming that w and σ are *a priori* independent, we get a standard Bayesian learning equation (see, for instance [Bun90])

$$Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y}) \;=\; \frac{L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) Pr(w) Pr(\sigma)}{\int_{w,\sigma} L(\boldsymbol{y} \mid \boldsymbol{x}, w, \sigma) Pr(w) Pr(\sigma) dw\, d\sigma} \;.$$

When using the network, the weights w are the primary concern and σ is of secondary interest, so we would also like to know $Pr(w \mid \boldsymbol{x}, \boldsymbol{y})$, which can be calculated from $\int_\sigma Pr(w, \sigma \mid \boldsymbol{x}, \boldsymbol{y}) d\sigma$. If we assume the standard non-informative prior for the standard error σ, $Pr(\sigma) = 1/\sigma$ [Ber85], the integration for σ given w can be done in closed form using the Γ integral [Ber85] (with a change of variables $\tau = 1/\sigma^2$). This yields

$$Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \;=\; \frac{L(\boldsymbol{y} \mid \boldsymbol{x}, w) Pr(w)}{\int_w L(\boldsymbol{y} \mid \boldsymbol{x}, w) Pr(w) dw} \;,$$

where

$$L(\boldsymbol{y} \mid \boldsymbol{x}, w) \;=\; \frac{\Gamma(\frac{N-1}{2})}{(N\pi)^{\frac{N-1}{2}} \sqrt{N}} \frac{1}{s^{N-1}} \;.$$

Furthermore, the expected value of σ^2 given w and $\boldsymbol{x}, \boldsymbol{y}$ can be calculated using the same integral, giving a standard result in statistics:

$$\mathbf{E}_{\sigma \mid w, \mathbf{x}, \mathbf{y}} \left(\sigma^2 \right) \;=\; \frac{N}{N-1} s^2 \;. \tag{1.3}$$

A similar calculation can be done for the Laplacian error model with unknown standard deviation Δ, and this time

$$\mathbf{E}_{\Delta \mid w, \mathbf{x}, \mathbf{y}} \left(\Delta^2 \right) \;=\; \frac{1}{N(N-1)} \left(\sum_{i=1,\ldots,N} |y_i - o(\boldsymbol{x}_i, w)| \right)^2 \;.$$

To summarize, posterior probabilities for a set of network weights w are as follows.

Lemma 1.5.1 *In the classification and regression frameworks described above for neural networks, assume the input $\boldsymbol{x}$ is a priori independent of the network weights w and other parameters such as σ and Δ, that is $Pr(\boldsymbol{x} \mid w, \sigma) = Pr(\boldsymbol{x})$. Posterior probabilities of network weights are as follows. For regression with Gaussian error and unknown σ,*

$$Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \;\propto\; Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) \;=\; Pr(w) \frac{\Gamma(\frac{N-1}{2})}{(N\pi)^{\frac{N-1}{2}} \sqrt{N} \left(\sum_{i=1,\ldots,N} (y_i - o(\boldsymbol{x}_i, w))^2 \right)^{\frac{N-1}{2}}} \;,$$

for regression with Laplacian error and unknown Δ,

$$Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \;\propto\; Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) \;=\; Pr(w) \frac{\Gamma(N)}{2^N \left(\sum_{i=1,\ldots,N} |y_i - o(\boldsymbol{x}_i, w)| \right)^N} \;,$$

and for classification

$$Pr(w \mid x, y) \propto Pr(w, y \mid x) = Pr(w) \prod_{i=1,\ldots,N} o_{y_i}(x_i, w) .$$

These posterior probabilities define the Bayesian solution to the problem of training networks. With a Gaussian error model, high posterior weights w trade-off the prior $Pr(w)$ and the mean-square error. With a Laplacian error model, the trade-off is with the average absolute deviation. In all cases, as the sample size N gets large, the prior term will become negilible and can be assumed constant.

1.6 Analyzing weights

The section describes how we can use the posterior analysis just given to assist basic tasks done during network training. In Section 1.6.1 we describes how we derive "cost functions" for a given set of weights from the posterior formulae and Section 1.6.2 describes how we can evaluate a set of weights. The cost function allows us to find a locally optimum set of weights and the evaluation allows us to compare the quality of one local optimum with another.

1.6.1 Cost functions

The posterior probabilities just given represent functions that should be maximized. Calculation is usually done in logarithms to prevent arithmetic underflow and to turn products into sums. Hence when searching for a high posterior set of weights we would like to minimize the "cost function" $-\log Pr(w \mid x, y)$. Notice that

$$-\log Pr(w \mid x, y) = -\log Pr(w, y \mid x) + \log Pr(y \mid x) ,$$

and since the term $\log Pr(y \mid x)$ is constant in w and difficult to calculate in general, we can instead minimize the cost function

$$\text{Cost}(w) = -\log Pr(w, y \mid x) ,$$

which can be directly computed using Lemma 1.5.1.

These cost functions differ from maximum likelihood methods for network training [BW87, EJM90] in that they introduce a prior term and sometimes have "nuisance" parameters eliminated. For instance, σ is termed a "nuisance" parameter when trying to determine a good set of weights and have no concern for the typical error. The prior term corresponds to the regularizing term found in smoothing methods [HT90] and used to good success in techniques such as weight-elimination [WRH91]. A similar Bayesian formulation is given in [Mac91].

Using the prior discussed in the previous section, with maximum *a posterior* analysis we would minimize in the regression case with Gaussian error and unknown σ

$$\begin{aligned} &-\log Pr(w, y \mid x) \\ &\quad = \frac{N-1}{2} \log \sum_{i=1,\ldots,N} (y_i - o(x_i, w))^2 - \log Pr(w) + \text{constant} , \end{aligned} \tag{1.4}$$

where the logarithm of the prior is determined as before. If the value of σ is known the cost to be minimized is

$$\frac{1}{2\sigma^2} \sum_{i=1,\ldots,N} (y_i - o(x_i, w))^2 - \log Pr(w) + \text{constant} .$$

Notice the difference between these above two cost functions. In the second case, the cost function is proportional to the mean squared error plus a "regularizing" term, but in the first case the logarithm of the mean squared error is taken. If the value of σ is determined from the inputs as well, as $o'(x_i, r)$, the cost to be minimized is now $-\log Pr(w, r, \boldsymbol{y} \mid \boldsymbol{x})$ given by

$$\sum_{i=1,\ldots,N} \frac{1}{2o'(x_i, r)^2}(y_i - o(x_i, w))^2 + \sum_{i=1,\ldots,N} \log o'(x_i, r) - \log Pr(r, w) + \text{constant} .$$

Minimization has to be done for r and w concurrently. A prior for r was given at the end of Section 1.5.3.

In the classification case, the overall cost is given by

$$-\log Pr(w, \boldsymbol{y} \mid \boldsymbol{x}) \;=\; -\sum_{i=1,\ldots,N} \log o_{y_i}(x_i, w) - \log Pr(w) + \text{constant} . \qquad (1.5)$$

If the output variable is boolean and represented as 1 for true and 0 otherwise, then the first sum expands to

$$-\sum_{i=1,\ldots,N} (y_i \log o_1(x_i, w) \;+\; (1 - y_i) \log(1 - o_1(x_i, w))) \quad .$$

Notice this is the familiar cross entropy error for the sample.

While this gives the cost function to minimize, we do not consider here the computational problem of how the minimization should be done. Various gradient descent, conjugate gradient or approximate second-order methods [BL88, EJM90] can be tried to solve the minimization problem. See, for instance, [PFTV88, Chap. 14] for a tutorial discussion of methods for solving the minimization problems for linear and non-linear regression with Gaussian and other more robust error models.

Notice the formulation given here assumes learning is done in batch (or epoch) mode because the posterior needs to be calculated on the full sample. When initializing search, it may help to use smaller batches to get somewhere near a local minimum, and perhaps do batch learning on the full sample when refining an already reasonable solution. This corresponds to the stochastic gradient descent strategy that is common in back-propagation implementations. We do not consider which strategy is better here.

1.6.2 Weight evaluation

To obtain a measure of the quality of each local maximum *a posterior* estimate $\widehat{w}$ found during search, an estimate of the local area under the posterior around $\widehat{w}$ is usually done. The actual posterior value itself is not the best measure of quality because some peaks may be thinner than others so they contain much less of the posterior probability in their vicinity. For instance, consider an idealized learning problem where the scalar parameter a is being learnt. Suppose a has posterior given in Figure 1.2. Notice the left peak is higher

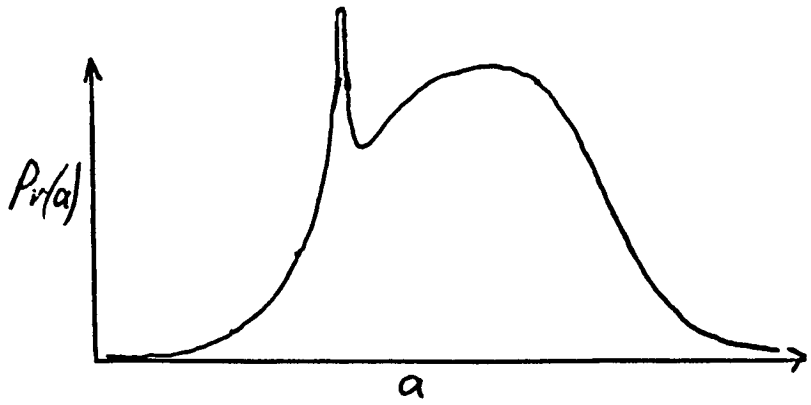

Figure 1.2: Posterior for a

but much thinner. The expected value of a, $\mathbf{E}_a(a)$, and other functions of a would therefore come more from the fatter peak on the right. This becomes more pronounced in higher dimensions, or when comparing networks of different dimension.

This local area estimate of quality lets different local maximum *a posterior* estimates be compared on an equal footing, for instance networks with different numbers of layers or connections. This local area estimate and successively coarser approximations to it corresponds to the various encoding measures such as MML [WF87] and MDL [Ris87], as discussed in the next section.

To make this estimate, we approximate the posterior at the local maximum $\widehat{w}$ by a function of the form

$$\begin{aligned} & Pr(w \mid x, y) \\ & \approx \quad Pr(\widehat{w} \mid x, y) \exp\left(-\frac{1}{2}(w-\widehat{w})^T I(\widehat{w})(w-\widehat{w}) + O((w-\widehat{w})^3)\right) , \end{aligned} \tag{1.6}$$

where

$$\begin{aligned} I(w) &= -\frac{\mathrm{d}^2 \log Pr(w \mid x, y)}{\mathrm{d}w\mathrm{d}w} = -\frac{\mathrm{d}^2 \log Pr(w, y \mid x)}{\mathrm{d}w\mathrm{d}w} , \\ &= \frac{\mathrm{d}^2 \mathrm{Cost}(w)}{\mathrm{d}w\mathrm{d}w} \end{aligned}$$

Notice the differentials are $\mathrm{d}w$ rather than ∂w because the derivative represents the full matrix of second derivatives w.r.t. the different weights $w_{n,m}$. Roughly, this approximation is valid for large sample size N because the posterior is found by normalizing for w a formula of the form $g(w)^N$ (see for instance [Ber85, p224]). The second derivative $I(\widehat{w})$ is taken of the cost function, the log posteriors such as in Equation (1.4) or (1.5). Generally, the sample size N should be at least some factor of the number of weights in the network.

When using a uniform prior, $I(\widehat{w})$ is referred to as the observed Fisher information matrix and its determinant is referred to as the Fisher information. Both play a central part in statistical theory [Ame85, Ber85]. In some simple non-network models, such as distributions

from the exponential family [Ber85], class probability trees and Bayesian networks on discrete variables [Bun90], the posterior can be dealt with exactly so the approximation using $I(\widehat{w})$ is not necessary. Because the constants in Equations (1.4) and (1.5) are independent of w, they can be ignored when evaluating the derivative. The local posterior area for w near $\widehat{w}$ is found by integrating out w using the multivariate Gaussian approximation, and is given by

$$Pr(near\ \widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \approx Pr(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \frac{(2\pi)^{|w|/2}}{\det(I(\widehat{w}))^{1/2}} ,$$

where $|w|$ is the dimensionality of the variable set w and $\det(\cdot)$ is the matrix determinant.

The desired quality measure is now given by the negative logarithm of the local area (and adding in the constant $-\log Pr(\boldsymbol{y} \mid \boldsymbol{x})$)

$$\begin{aligned} \text{Eval}(\widehat{w}) &= -\log Pr(near\ \widehat{w}, \boldsymbol{y} \mid \boldsymbol{x}) \\ &= -\log Pr(\boldsymbol{y} \mid \widehat{w}, \boldsymbol{x}) - \log Pr(\widehat{w}) - \frac{|w|}{2}\log(2\pi) + \frac{1}{2}\log\det(I(\widehat{w})) , \quad (1.7) \end{aligned}$$

which wants to be minimized. Notice the first and second terms together give $\text{Cost}(\widehat{w})$. We separate them out here because they correspond to the likelihood and prior components respectively.

Notice also, that from the functional form of Equation (1.6), the matrix inverse of $I(\widehat{w})$, $I(\widehat{w})^{-1}$, represents an approximate posterior variance-covariance matrix for w, [Ber85, p224], and is at least approximately correct in the neighborhood of $\widehat{w}$. This means, for instance, that the posterior variance of the weight $w_{n,m}$ in the neighborhood of $\widehat{w}$, denoted $\mathbf{v}_{w|x,y,near\ \widehat{w}}(w_{n,m})$, is given by the diagonal entry for $w_{n,m}$ in the matrix $I(\widehat{w})^{-1}$. That is, if $\mathbf{v}_{w|x,y,near\ \widehat{w}}(w_{n,m})$ is large, then the weight $w_{n,m}$ is poorly determined by the data and perhaps should be ignored.

The determinant $\det(I(\widehat{w}))$ could be approximated by looking only at the diagonals or block diagonals (block corresponds to one node which is square in the number of weights for the node). Individual second derivatives can be determined using a number of methods [BW91b]. Fortunately, this only has to be done once for each local minima $\widehat{w}$ found, instead of repeatedly during search, so a fast approximation is not essential. Fast packages exist, such as the Fortran MATLIB, that can find determinants and inverses of matrices of row/column size several hundred in a few minutes.

1.6.3 Minimum encoding methods

Quality measures similar to $\text{Eval}(\widehat{w})$ can be derived using various quantitization, encoding and approximation methods. We discuss these briefly here to draw the strong connections between $\text{Eval}(\widehat{w})$ and the often discussed encoding measures given in [WF87, BC91, Ris87]. We view these methods as potentially useful approximations but do not consider their application here.

Rissanen's MDL

Rissanen develops his MDL measure [Ris87] as an upper bound on $-\log Pr(w, \boldsymbol{y} \mid \boldsymbol{x})$. For exposition purposes, we develop a related bound here. $I(\widehat{w})$ is a positive indefinite symmetric

matrix (since $\widehat{w}$ is a local maxima), so[1]

$$\det I(\widehat{w}) \;\leq\; \left(\frac{\text{trace}I(\widehat{w})}{|w|}\right)^{|w|} ,$$

where trace$A$ is the sum of the diagonal entries in the matrix A. Now for large N, trace$I(\widehat{w})/N$ is a mean so is approximately Gaussian, with mean $O(|w|)$ and standard deviation $O(|w|/\sqrt{N})$. Therefore with probability approaching 1 as N gets larger,

$$\frac{1}{2}\log\det I(\widehat{w}) \;=\; \frac{|w|}{2}\log N + O(|w|) .$$

We get that

$$\text{Eval}(\widehat{w}) \;=\; -\log Pr(\boldsymbol{y} \mid \widehat{w}, \boldsymbol{x}) - \log Pr(\widehat{w}) + \frac{|w|}{2}\log N + O(|w|) .$$

This is an approximation developed by Schwarz and others [BC91]. The corresponding MDL form ignores the term $\log Pr(\widehat{w})$ which is probably $O(|w|)$ anyway and has an additional term $\frac{|w|}{2}\log|w|$. In our case, $O(|w|)$ is quite large because we are dealing with networks with many weights. It is not unusual in practice for the number of weights to be a similar order of magnitude to sample size N, so ignoring terms of order $O(|w|)$, as this approximation does, is probably unwise. We conclude this approximation is informative but perhaps too crude.

Wallace *et al.* and Barron *et al.*'s measure

Wallace and Freeman [WF87] and Barron and Cover [BC91] interpret a form like Eval$(\widehat{w})$ as the cost of encoding y and $\widehat{w}$ given $\boldsymbol{x}$. Wallace *et al.* refer to this as the minimum message length (MML). Of course, $\boldsymbol{y}$ and $\widehat{w}$ can only be encoded to finite precision (in classification, $\boldsymbol{y}$ is finite already). The precision of $\boldsymbol{y}$ is implicit in the supplied data, so if the volume of the precision for the data vector $\boldsymbol{y}$ is δ then the cost of encoding $\boldsymbol{y}$ given $\widehat{w}$ and $\boldsymbol{x}$ is

$$-\log Pr(\boldsymbol{y} \mid \widehat{w}, \boldsymbol{x}) - \log\delta .$$

Because $\boldsymbol{y}$ is encoded given $\widehat{w}$, we must encode $\widehat{w}$ without knowledge of $\boldsymbol{y}$. The second, third and fourth terms of Equation (1.7) are not an appropriate code length, however, because $I(\widehat{w})$ is dependent on $\boldsymbol{y}$.

The third and fourth terms of Equation (1.7) essentially correspond to the "precision" of $\widehat{w}$ because if the continuous space for w is quantified then the cell in the neighborhood of $\widehat{w}$ has prior probability given approximately by $Pr(\widehat{w})\text{Area}(\widehat{w})$ where Area$(\widehat{w})$ is the area of the cell, so $-\log\text{Area}(\widehat{w})$ is the precision to which $\widehat{w}$ is specified. To interpret this precision component, notice that if the weights w happened to be *a posterior* independent (an unlikely event), then

$$-\frac{|w|}{2}\log(2\pi) + \frac{1}{2}\log\det(I(\widehat{w})) \;\approx\; -\sum_{n,m}\log\left(2.51\cdot\sqrt{\mathrm{v}_{w|\mathrm{x},\mathrm{y},near\ \widehat{w}}(w_{n,m})}\right) .$$

[1] The determinant is a product of eigenvalues, whereas the trace is a sum of eigenvalues, which in this case are all non-negative. The inequality follows by maximizing $\prod_i \lambda_i$ given a fixed value for $\sum_i \lambda_i$.

The recommended precision for encoding each weight is therefore approximately 2.5 times the posterior standard deviation of the weight.

Wallace *et al.* and Barron *et al.* make the fourth term in Equation (1.7) independent of y using an involved quantification argument. We can interpret this by noting

$$\frac{1}{2}\log\det(I(\widehat{w})) \;=\; \frac{|w|}{2}\log N + \frac{1}{2}\log\det\left(\frac{1}{N}I(\widehat{w})\right) \;.$$

The term $\frac{1}{N}I(\widehat{w})$ is now in the form of an average since in general $I(\widehat{w})$ is a sum across the N points in the sample plus a fixed prior term. We now replace this "average" observed Fisher information matrix by what is called the expected Fisher information matrix:

$$\begin{aligned}\text{Coding}(y,\widehat{w}) \;&=\; -\log Pr(y \mid \widehat{w},x) - \log\delta - \log Pr(\widehat{w}) - \frac{|w|}{2}\log(2\pi) - \frac{|w|}{2} \\ &\quad + \frac{|w|}{2}\log N + \frac{1}{2}\log\det(\overline{I}(\widehat{w}))\;,\end{aligned}$$

where $\overline{I}(w)$ is the expected Fisher information matrix (the expected value of the observed Fisher information for a single pattern) given by[2]

$$\begin{aligned}\overline{I}(w) \;&=\; \mathbf{E}_{x,y|w}\left(-\frac{\mathrm{d}^2\log Pr(x,y \mid w)}{\mathrm{d}w\mathrm{d}w}\right) \;, \\ &=\; \mathbf{E}_{x}\left(\mathbf{E}_{y|w,x}\left(-\frac{\mathrm{d}^2\log Pr(y \mid w,x)}{\mathrm{d}w\mathrm{d}w}\right)\right) \;, \\ &\approx\; \frac{1}{N}\sum_{i=1,\ldots,N}\mathbf{E}_{y|w,x_i}\left(-\frac{\mathrm{d}^2\log Pr(y \mid w,x_i)}{\mathrm{d}w\mathrm{d}w}\right) \;,\end{aligned}$$

where (x,y) in these formula denotes a single pattern rather than the training sample $(\boldsymbol{x},\boldsymbol{y})$. Notice that this last formula is an estimate of $\frac{1}{N}I(\widehat{w})$ involving knowledge of the sample inputs $\boldsymbol{x}$ but not the sample outputs y. It is therefore a valid inclusion in the code-length of w.

1.7 Applications to network training

This section describes some applications of the tools just developed.

1.7.1 Weight variance and elimination

The Gaussian approximation to the likelihood near a local maximum *a posterior* also gives a statistically sound method for pruning of networks [LDS90]. The matrix inverse of $I(\widehat{w})$, $I(\widehat{w})^{-1}$, represents an approximate posterior variance-covariance matrix for w, so we can use this to test if a weight is significantly different from zero. Those that are not can be set to zero.

[2] There is actually some slight of hand here. Wallace *et al.* and Barron *et al.* develop their framework to deal with likelihoods of the form $p(x|w)$ for data x and parameters w whereas we are considering conditional likelihoods such as $p(y|w,x)$. Here we give our interpretation of their methods in this different context.

If we use the diagonal approximation to this matrix, then we get an estimate of the posterior variance of the weight $w_{n,m}$ in the neighborhood of $\widehat{w}$

$$\mathrm{v}_{w|\mathbf{x},\mathbf{y}}(w_{n,m}) \approx \mathrm{v}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(w_{n,m}) \approx -1 \Big/ \left.\frac{\partial \log Pr(w \mid \boldsymbol{x}, \boldsymbol{y})}{\partial w_{n,m} \partial w_{n,m}}\right|_{w=\widehat{w}} .$$

Because $\widehat{w}$ is a local maximum for the posterior, the second derivative must be non-positive so the variance estimate non-negative. A zero variance estimate means some other estimate will have to be made. A large variance means the sample gave us little idea as to what value the weight should be. So if the magnitude of a weight is within a standard deviation (square root of the variance), the weight can be safely set to zero, which is similar to the suggestion in [Ish90]. If the standard deviation of a weight is small, and the weight itself $w_{n,m}$ is smaller than the standard deviation, then the weight can be safely set to zero. Alternatively, if the weight's standard deviation is large but the magnitude of the weight larger, then the variance gives no support for zeroing the weight.

A more thorough test goes as follows. The quadratic term $(w-\widehat{w})^T I(\widehat{w})(w-\widehat{w})$ appears in the multivariate Gaussian approximation for the posterior for the weights w. For large N, we therefore get that this quadratic term is approximately χ-squared with $|w|$ degrees of freedom. The $X\%$ confidence region (for instance, $X = 99.0$) for the weights w is the set of weights within the upper $X\%$ point of the $\chi^2_{|w|}$ distribution, $\chi^2_{|w|,X}$, giving

$$w \quad : \quad (w-\widehat{w})^T I(\widehat{w})(w-\widehat{w}) \leq \chi^2_{|w|,X} \quad .$$

For instance, the upper 99% point for 30 weights is given from standard χ-squared tables as 50.89. We therefore proceed as follows. We repeatedly set weights to zero while the resultant weights remain in the $X\%$ confidence region. Notice this simple test can be done directly from $I(\widehat{w})$ without having to first calculate a determinant or inverse, so might also be done at stages during the back-propagation cycle.

1.7.2 Prediction and generalization error

Once search has located some local minima $\widehat{w}$ to the cost function, they can be used in inference on new patterns. A naive approximation is to say that w must now be $\widehat{w}$ and to make inference about y and σ^2, etc., based entirely on this local minima $\widehat{w}$ using $y = o(x', \widehat{w})$, etc. The reason this is naive is that in reality we cannot be sure that the "true" or "optimum" w is $\widehat{w}$. For instance, with a sample twice the size we might change our estimate of w quite dramatically. A full Bayesian approach says to take into account the many other values of w near $\widehat{w}$ or even some other local minima because these other values just might happen to be the "true" one. Only when we have a really large sample (for instance, when uniform convergence bounds tell us the sample is large enough [Hau91]) can we be sure that $\widehat{w}$ is a sufficiently good estimate so that no other need be considered.

In principle what we would like to calculate for a new pattern x' is $\mathrm{E}_{w|\mathbf{x},\mathbf{y}}(o(x', w))$ and $\mathrm{E}_{w,\sigma|\mathbf{x},\mathbf{y}}(\sigma^2)$, etc. These values give us the posterior average of the quantities $o(x', w)$ and σ^2 that give better predictions for these quantities on average than those calculated for $w = \widehat{w}$. The problem here is that we cannot calculate the posterior $Pr(w \mid \boldsymbol{x}, \boldsymbol{y})$ in reasonable time, and only have the Gaussian approximation to the posterior described above.

This section describes how these predictions can be approximated using the posterior approximations available. These approximations all make use of the second derivatives of the cost function and the approximate weight variance discussed above. While it is hard to

predict how accurate these approximations will be in practice, their form at least gives some idea of the interaction between weight variances and the behavior of the network output for w in the vicinity of $\widehat{w}$.

Predictions are given for three quantities. The first is an approximation to $\mathbf{E}_{w|\mathbf{x},\mathbf{y}}\left(o(x',w)\right)$. In classification this can be used to get an estimate of the posterior distribution for y given x', and can be used to predict generalization error, and in regression this gives an estimate of the posterior mean of y. This can also be used to give generalization error for classification. The second prediction is for $\mathbf{v}_{w|\mathbf{x},\mathbf{y}}\left(o(x',w)\right)$ which measures our uncertainty in the first prediction. Notice as the sample size gets large this uncertainty will shrink to zero. We give two versions of this uncertainty depending on which approximation to $\mathbf{E}_{w|\mathbf{x},\mathbf{y}}\left(o(x',w)\right)$ is used. The third prediction is for $\mathbf{E}_{w,\sigma|\mathbf{x},\mathbf{y}}\left(\sigma^2\right)$ which predicts generalization error for regression with unknown standard deviation.

Predictions for a local minima

In what follows, we use the notation $\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(U(w))$ to denote the expectation of the quantity $U(w)$ when averaged according to the approximate Gaussian posterior for w in the vicinity of $\widehat{w}$. That is

$$\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(U(w)) \;=\; \int_w U(w)\,\frac{\det(I(\widehat{w}))^{\frac{1}{2}}}{(2\pi)^{|w|/2}}\exp\left(-\frac{1}{2}(w-\widehat{w})^T I(\widehat{w})(w-\widehat{w})\right)dw\ .$$

We evaluate these integrals by using standard moments of the multivariate Gaussian [Ber85], and approximating $U(w)$ in the vicinity of $\widehat{w}$ using the second-order Taylor expansion which in vector notation is as follows:

$$\begin{aligned} U(w) \;\approx\; & U(\widehat{w}) \;+\; (w-\widehat{w})^T\cdot\left.\frac{\mathrm{d}U(w)}{\mathrm{d}w}\right|_{w=\widehat{w}} \\ & +\;\frac{1}{2}\;(w-\widehat{w})^T\cdot\left.\frac{\mathrm{d}^2U(w)}{\mathrm{d}w\mathrm{d}w}\right|_{w=\widehat{w}}\cdot(w-\widehat{w})\ . \end{aligned}$$

Again, $\frac{\mathrm{d}o(x',w)}{\mathrm{d}w}$ denotes a vector of first derivatives and the second derivative a matrix. This gives

$$\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(U(w)) \;\approx\; U(\widehat{w}) \;+\; \frac{1}{2}\mathrm{trace}\left(I(\widehat{w})^{-1}\cdot\left.\frac{\mathrm{d}^2U(w)}{\mathrm{d}w\mathrm{d}w}\right|_{w=\widehat{w}}\right)\quad ,$$

where, again, trace$A$ denotes the sum of the diagonal entries of the matrix A. If any of the approximate weight variances are large, then the approximation of $\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(U(w))$ will require accurate estimates of $U(w)$ for w far from $\widehat{w}$. In this case the Taylor expansion will be a poor approximation and our approximate predictions will be poor. Notice we could also use the diagonal approximation for the matrices $I(\widehat{w})$ and the second derivatives of $U(w)$, however, the estimates may then become very poor and only be useful to indicate where potential problems lie. These and other approximations are discussed in [Pre89].

$\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o(x',w))$: the posterior expected output of the network given input x', averaged over weights in the vicinity of $\widehat{w}$. This is approximated by

$$o(x',\widehat{w}) + \frac{1}{2}\mathrm{trace}\left(I(\widehat{w})^{-1}\left.\frac{\mathrm{d}^2o(x',w)}{\mathrm{d}w\mathrm{d}w}\right|_{w=\widehat{w}}\right)\quad .$$

This modifies the output $o(x', \widehat{w})$ to account for how the output $o(x', w)$ varies in the neighborhood of $\widehat{w}$, tempered by the posterior variance-covariance of w. In the case of classification, this gives an estimate of out-of-sample accuracy for the network given by

$$\frac{1}{N}\sum_{i=1}^{N} \mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o_{y_i}(x_i, w)) \approx \frac{1}{N}\sum_{i=1}^{N} o_{y_i}(x_i, \widehat{w}) + \frac{1}{2N}\mathrm{trace}\left(I(\widehat{w})^{-1} \left.\frac{\mathrm{d}^2 \sum_{i=1}^{N} o_{y_i}(x_i, w)}{\mathrm{d}w\mathrm{d}w}\right|_{w=\widehat{w}} \right) \quad . \quad (1.8)$$

$\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}\left((o(x', w) - o(x', \widehat{w}))^2\right)$: when using $o(x', \widehat{w})$ to estimate $o(x', w)$, the expected variance of the output in the neighborhood of $\widehat{w}$, which gives a measure of variance for our posterior uncertainty when using $o(x', \widehat{w})$ to estimate $o(x', w)$. For instance, in the regression case $o(x', \widehat{w})$ corresponds to an estimate of the mean value of y conditioned on x'. The variance is approximated by

$$\left.\frac{\mathrm{d}o(x', w)}{\mathrm{d}w}\right|^{T}_{w=\widehat{w}} \cdot I(\widehat{w})^{-1} \cdot \left.\frac{\mathrm{d}o(x', w)}{\mathrm{d}w}\right|_{w=\widehat{w}} \quad (1.9)$$

$\mathbf{v}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o(x', w))$: the variance of the estimate $\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o(x', w))$ in the neighborhood of $\widehat{w}$. This is approximated by

$$\left.\frac{\mathrm{d}o(x', w)}{\mathrm{d}w}\right|^{T}_{w=\widehat{w}} \cdot I(\widehat{w})^{-1} \cdot \left.\frac{\mathrm{d}o(x', w)}{\mathrm{d}w}\right|_{w=\widehat{w}} - \frac{1}{4}\mathrm{trace}\left(I(\widehat{w})^{-1} \cdot \left.\frac{\mathrm{d}^2 o(x', w)}{\mathrm{d}w\mathrm{d}w}\right|_{w=\widehat{w}} \right) \quad .$$

Notice this is lower than the previous variance estimate of Formula (1.9) due to the second term which corrects for the fact that we are now using a better estimate for $o(x', w)$.

$\mathbf{E}_{w,\sigma|\mathbf{x},\mathbf{y},near\ \widehat{w}}(\sigma^2)$: the posterior expected value of σ^2 in the regression case with Gaussian error and unknown σ. This is sometimes termed the generalization error. This is approximated using Equation (1.3) by

$$\frac{1}{N-1}\sum_{i=1,\ldots,N} (\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o(x_i, w)) - y_i)^2 + \frac{1}{N-1}\sum_{i=1,\ldots,N} \mathbf{v}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o(x_i, w)) \ ,$$

where the variances in the second sum were just given. This simplifies to

$$\frac{N}{N-1}\, s^2\big|_{w=\widehat{w}} + \frac{N}{2(N-1)} trace\left(I(\widehat{w})^{-1} \cdot \left.\frac{\mathrm{d}^2 s^2}{\mathrm{d}w\mathrm{d}w}\right|_{w=\widehat{w}} \right) \quad . \quad (1.10)$$

Notice the first term is the observed variance, and the second term increases this because the estimate $\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(o(x_i, w))$ is only approximate and has a variance of its own.

Multiple networks

If multiple local minima $\widehat{w}$ of similar quality have been found during restarts in the search for weights, then the expected values above can be averaged over the different local minima $\widehat{w}$ to produce a pooled estimate. This same "multiple models" approach can produce significant improvement in out-of-sample prediction when learning classification trees [Bun90]. It corresponds to a Monte Carlo estimation of the full posterior for w, rather than just an approximation about $\widehat{w}$. Suppose a small set $\widehat{W}$ of local minima are found, and for each we have a proportion $p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y})$ to be used in the Monte Carlo estimation below, where $\sum_{\widehat{w} \in \widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) = 1$. For instance, these proportions could be constructed proportional to $Pr(near\ \widehat{w}, \boldsymbol{y} \mid \boldsymbol{x})$, although equal proportions are sometimes used in Monte Carlo estimation. The estimate of the posterior for the weights w is now a weighted some of the Gaussians in the neighborhood of each $\widehat{w}$:

$$Pr(w \mid \boldsymbol{x}, \boldsymbol{y}) \approx \sum_{\widehat{w} \in \widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) Pr(w \mid \boldsymbol{x}, \boldsymbol{y}, near\ \widehat{w}),$$

where the Gaussian approximation described at the beginning of this section is used for each $Pr(w \mid \boldsymbol{x}, \boldsymbol{y}, near\ \widehat{w})$.

This leads to the following corrections for the previous predictions. To find the expected output of the network for different weights, we pool the expected outputs for the individual local minima in $\widehat{W}$:

$$\mathbf{E}_{w|\mathbf{x},\mathbf{y}}\left(o(\boldsymbol{x}', w)\right) \approx \sum_{\widehat{w} \in \widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}\left(o(\boldsymbol{x}', w)\right) \quad .$$

To find the variance of this value, we pool the variances for individual local minima together with a measure of how much the output varies from minima to minima:

$$\begin{aligned} \mathbf{v}_{w|\mathbf{x},\mathbf{y}}\left(o(\boldsymbol{x}', w)\right) \approx & \sum_{\widehat{w} \in \widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \mathbf{v}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}\left(o(\boldsymbol{x}', w)\right) \\ & + \mathbf{v}_{\widehat{w}|\mathbf{x},\mathbf{y}}\left(\mathbf{E}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}\left(o(\boldsymbol{x}', w)\right)\right) \quad . \end{aligned}$$

To estimate the standard error σ in regression with a Gaussian error model, we pool the individual estimates of σ.

$$\mathbf{E}_{w,\sigma|\mathbf{x},\mathbf{y}}\left(\sigma^2\right) \approx \sum_{\widehat{w} \in \widehat{W}} p(\widehat{w} \mid \boldsymbol{x}, \boldsymbol{y}) \mathbf{E}_{w,\sigma|\boldsymbol{x},\boldsymbol{y},near\ \widehat{w}}\left(\sigma^2\right) \ .$$

1.7.3 Adjustments for missing values

In many practical problems, training patterns are supplied with some values of variables missing, unknown or undetermined. Of course if a pattern has its output value missing the pattern can just be ignored. If an input variable has a value missing, however, we need to deal with it. Approximate methods for dealing with this kind of problem exist when learning tree classifiers [Qui89]:

- One can ignore the training pattern with missing values. Stochastic learning can proceed initially using only the patterns without missing values.

- One can replace the missing value by some simple estimate such as the modal or mean value. For instance, if the boolean input variable x_i is either 0 (false) or 1 (true) and if it is unknown, set it to be about 0.5.
- One can complete the pattern in various ways (to fill in missing values) and treat each of these completions as a partial pattern (so the sum of fractional patterns adds to 1). We explain this idea in the context of feed-forward networks below. Tree learning algorithms do this completion in an efficient demand driven manner (i.e. they complete a missing value only when the value is asked for by the algorithm) that unfortunately is not available with feed-forward networks.

The second approach works quite well and is the simplest to implement, however the third approach gives the best performance in terms of out-of-sample prediction.

In this section we develop a modified algorithm for handling missing values related to the third approach above. It also approximates the Bayesian normative approach for handling missing values because we derive the approach here using standard laws of probability. The method is another example of the use of mixture models [JJNH91] used for adaptive mixtures of local experts, and the equations derived have a related form. We do the analysis for the regression problem with Gaussian error and unknown standard deviation. The other cases are similar. We assume we have a model of some kind denoted F that gives rough prediction of the missing values from the known values. The model F might correspond to simple linear models or some other form readily calculated from the data.

The training sample consists of input vectors $x_1, \ldots, x_N$ and corresponding output values $y_1, \ldots, y_N$. But in this case some of the input vectors x_i have missing values. Let $\mathit{unknown}(x_i)$ denote the set of variables in the i-th input vector x_i that have missing values. For instance, given the i-th boolean pattern $x_i = (1, 0, ?, 0, ?, ?)$ where ? denotes a missing value, then $\mathit{unknown}(x_i)$ is the 3rd, 5th and 6th variables. Given an assignment to these, $x' \in \mathit{unknown}(x_i)$, let $x_i x'$ denote one possible completion for the input vector x_i. Then the likelihood of the pattern is now given by

$$l(y_i \mid x_i, w, \sigma, F) \;=\; \mathrm{E}_{x' \in \mathit{unknown}(x_i) \mid x_i, F}\left(l(y \mid x_i x', w, \sigma)\right)$$

where the expectation is done using the model F for predicting the unknown values. We can approximate this stochastically by selecting a few possible completions $\mathit{compl}(x_i)$ and weighting them to give

$$l(y_i \mid x_i, w, \sigma, F) \;\approx \sum_{x' \in \mathit{compl}(x_i)} p(x' \mid x_i, F) l(y \mid x_i x', w, \sigma)\,, \tag{1.11}$$

where the weights $p(x' \mid x_i, F)$ indicate likelihood of the different completions based on the model F. This formula can now be evaluated because each likelihood $l(y \mid x_i x', w, \sigma)$ can be determined as in Section 1.5.1. For instance, in the example above, we may select 3 completions according to the model for predicting missing values as

$$\mathit{compl}(x_i) = \{\,(1,0,0,0,0,1), (1,0,0,0,1,1), (1,0,1,0,1,0)\,\}$$

and give them equal weighting of 1/3 each. Completions and weightings can be determined once before the network training begins. Notice the crudest estimate is to use only the single modal or mean value for the missing values with a weight of 1, as is often done.

Network training now consists of using these modified likelihoods. In the regression case we are considering, the standard deviation can no longer be marginalized out so the cost

function is now $-\log Pr(w, \sigma, \boldsymbol{y}|F, \boldsymbol{x})$. Suppose the patterns in *missing* have missing values and those in *complete* have all values given, then the cost is:

$$\sum_{i \in complete} \frac{1}{2\sigma^2}(y_i - o(x_i, w))^2 + (N+1)\log\sigma - \log Pr(w) + \text{constant}$$

$$- \sum_{i \in missing} \log\left(\sum_{x' \in compl(x_i)} p(x'|x_i, F) \exp\left(\frac{-1}{2\sigma^2}(y_i - o(x_i x', w))^2 \right) \right) .$$

While this looks more involved than the original cost function of Equation (1.4), its derivatives are not that different.

$$\begin{aligned} &\frac{\partial \text{Cost(w)}}{\partial w_{n,m}} \\ &= -\frac{\partial \log Pr(w)}{\partial w_{n,m}} - \frac{1}{2\sigma^2} \sum_{i \in complete} (y_i - o(x_i, w)) \frac{\partial o(x_i, w)}{\partial w_{n,m}} \\ &\quad -\frac{1}{2\sigma^2} \sum_{i \in missing} \sum_{x' \in compl(x_i)} w(x' \mid x_i, F, \sigma) \cdot (y_i - o(x_i x', w)) \frac{\partial o(x_i x', w)}{\partial w_{n,m}} , \end{aligned} \tag{1.12}$$

where the proportions $w(x'|x_i, F, \sigma)$ are calculated as

$$w(x'|x_i, F, \sigma) \equiv \frac{p(x' \mid x_i, F) \exp\left(\frac{-1}{2\sigma^2}(y_i - o(x_i x', w))^2\right)}{\sum_{x' \in compl(x_i)} p(x' \mid x_i, F) \exp\left(\frac{-1}{2\sigma^2}(y_i - o(x_i x', w))^2\right)} .$$

Compare the two sums in Equation (1.12). In the second, the term for the completion $x_i x'$ appears weighted by the proportion $w(x'|x_i, F, \sigma)$ but in the term for the first sum each (already complete) pattern x_i is effectively weighted by 1. For this reason we say each completion participates as a partial pattern. Notice, also that the proportions $w(x'|x_i, F, \sigma)$ essentially pick out that completion x' giving lowest squared error. Since the derivative of $o(x_i x', w)$ w.r.t. the completion x' is available after the back-propagation step, we could dynamically alter the completions x' during each learning step so their mean-square error become lower.

The derivative of Cost w.r.t. σ is in a similar form but this time a fixed-point equation (σ also occurs on the right-hand side) for σ can be derived:

$$\sigma = \frac{1}{N+1} \left(\sum_{i \in complete} (y_i - o(x_i, w))^2 + \sum_{i \in missing} \sum_{x' \in compl(x_i)} w(x' \mid x_i, F, \sigma) \cdot (y_i - o(x_i x', w))^2 \right) .$$

The standard deviation σ can be updated iteratively along with the weights w in each cycle. Again we see the "partial patterns" alter the usual form for the variance. This is a common result in mixture models.

A disadvantage of this method is that each pattern with missing values now produces a number of completed patterns, each of which must be run through the network. For instance, if every pattern has missing values and c different completions are used for each, then computation is increased by a factor of c. One way around this would be to have the network first learn for patterns with no missing values, and then to fine tune by including the remaining patterns. Of course, the extra computation should give improved performance

as c is increased due to the normative justification of the approach. Also, the completed patterns can be dynamically altered during training with little extra overhead, to produce completions with lower mean-square error.

1.8 Conclusion

This paper has covered Bayesian theory relevant to the problem of training feed-forward connectionist networks. We now sketch out how this might be put together in practice, assuming a standard gradient descent algorithm as used during search.

For network training, the principle steps are as follows:

1. Choose an appropriate network structure and size based on prior knowledge about the application (see for instance the discussion in [LeC89] regarding choice of network), and select a prior on the weights. Notice a small number of different structures could be selected and the method will the best select the best.

2. Construct completions and their proportions $p(x'|x_i, F)$ for each training pattern with missing values. If stochastic training is used rather than epoch training, each set of completions should always be run through the network together so that the appropriate term in Equation (1.12) can be calculated.

3. Train to a local minima $\widehat{w}$ as per usual but incorporating the adjustments described for missing values in Section 1.7.3. Appropriate cost functions are given in Section 1.6.1. In principle, the weight variances, weights evaluation and predictions only apply if $\widehat{w}$ found is a true local minima to the cost function so epoch training might have to be used in the last few cycles to ensure this.

4. Possibly, use the weight elimination strategy of Section 1.7.1 to force some weights to zero, and continue training the network with forced weights remaining at zero.

5. Once a local minima is found, estimate the quality of the local minima by finding second derivatives for every training pattern and combining them in the Evaluation measure of Section 1.6.2. Calculation of second derivatives is described in [BW91b].

6. Do random restarts of the network to repeat the last three steps and find other local minima. The estimated weight variances $\mathbf{v}_{w|\mathbf{x},\mathbf{y},near\ \widehat{w}}(w_{n,m})$ or $I(\widehat{w})$ and the evaluation measure $\text{Eval}(\widehat{w})$ should be retained for each low cost local minima $\widehat{w}$ saved. If several different network structures are being tried, then repeat the last three steps for these as well.

7. Choose a few networks with the best evaluation ($\text{Eval}(\widehat{w})$). Notice that the networks or local minima should not be chosen on the basis of their generalization error (as given in Section 1.7.2) because the generalization error for a particular local minima $\widehat{w}$ is estimated based on the assumption that the network structure is "correct" and the "true" weights are in fact quite near $\widehat{w}$.

8. Estimate the generalization error (for out-of-sample prediction) using the Formulas (1.8) or (1.10), possibly combined using the pooled versions at the end of Section 1.7.2. Now that a high posterior structure and set of weights has been chosen, the assumptions behind the formula are reasonable.

Once some local minima have been found, inference can be done on new patterns. Relevant approximate predictions are described in Section 1.7.2 that apply if the weight variances are small. Standard inference does forward propagation of the inputs for the new pattern to obtain the output. The adjustments described involve approximation of output posterior variance of the output (error bars), better approximation of expected output and variance, and averaging over multiple local minima.

These adjustments to the standard back-propagation procedure increase computation during back-propagation in most cases by at most a factor. Subsequent inference such as calculating variances require matrix calculations that can be done using fast standard matrix packages. While these methods come with the normative backing of Bayesian statistics, implementation often reveals lessons on how the various approximations and optimizations could be better made. One important issue here is the quality of the Gaussian approximation to the posterior of the weights. Since the whole approach rests on this, more evaluation and experience is required. A smooth transition also needs to be developed between Bayesian and uniform convergence methods to handle those cases where training samples become larger.

Acknowledgements

One of us (ASW) would like to thank David Rumelhart for many fruitful discussions. The other (WLB) would like to thank Robin Hanson likewise.

Chapter 2

Calculating Second Derivatives on Feed-Forward Networks

Recent techniques for training connectionist feed-forward networks require the calculation of second derivatives to calculate error bars for weights and network outputs, and to eliminate weights, etc. This note describes some exact algorithms for calculating second derivatives. They require at the worst case approximately $2K$ back/forward-propagation cycles where K is the number of nodes in the network. For networks with two-hidden layers or less, computation can be much quicker. Three previous approximations, ignoring some components of the second derivative, numerical differentiation, and scoring, are also reviewed and compared.

2.1 Introduction

Recent improvements to back-propagation methods for training neural networks have considered several tasks: speeding up learning using more sophisticated gradient descent algorithms [BL88, EJM90]; eliminating insignificant weights in order to find networks of an optimal size [LDS90]; the placing of error bars on the weights (to determine which weights are poorly estimated from the data and are perhaps suitable for elimination) and on the network outputs [DL91, Mac91]; and comparing the "posterior" quality of weights in different networks trained on the same data [BW91a, Mac91], the MDL principle and related encoding techniques are generally considered to be approximations of these [BB88, BW91a]. These techniques share one requirement in common: they all require calculation of second derivatives of the energy function or network output w.r.t. the network weights. The first derivatives are of course calculated during standard back-propagation. Le Cun *et al.* have suggested an approximate calculation that ignores certain terms [LDS90] and MacKay, sometimes finding this too inaccurate, used a more costly but accurate method of numerical differentiation [LDS90, Mac91]. Second derivatives are used by MacKay and their use is suggested by Buntine and Weigend [BW91a] because they have a relationship with important quantities such as the posterior variance of the network weights, i.e. "error bars", and the description length of a set of weights used in evaluating the quality of the set of

weights. Because these quantities are only required at the end of network training, rather than during each training cycle, their calculation does not have to be very efficient. Also, some of the above methods require the complete set of second derivatives, some require only the diagonal entries, and some approximations use block diagonals or other subsets of the full set.

2.2 Notation

The networks we consider consist of a directed acyclic graph, giving the network structure, together with the activation functions at each node. A (directed) connection from node n to m represents that node m receives node n's activation during the inference stage, often multiplied with a weight $w_{m,n}$. Let K denote the number of nodes in the network. Let the set $\mathcal{A}$ denote the set of node pairs corresponding to a connection, so $(n,m) \in \mathcal{A}$ denotes that there is a connection from nodes n to m. Let $\mathcal{A}^*$ be such that $(n,m) \in \mathcal{A}^*$ if and only if there is some sequence of connections in $\mathcal{A}$ from n to m. The sequence may be null so that $(n,n) \in \mathcal{A}^*$ by default. This is referred to as the transitive closure of $\mathcal{A}$. For instance, if the network is layered and fully connected between layers, then $(n,m) \in \mathcal{A}^*$ will hold whenever n is in a lower numbered layer than m or n is equal to m. The activation of any node $n \in \mathcal{N}$ is denoted u_n which is a real number. The activation function for the node m is a function of the activations u_n for nodes n inputting to node m, and the function is parameterized by a vector of parameters w_m. The activation function for a node n can take the following forms (for some functions $\phi_{n,k}$ and f_n)

$$u_n = f_n(v_n, w_{n,0}) \quad \text{where} \quad v_n \equiv \sum_{(k,n)\in\mathcal{A}} \phi_{n,k}(u_k, w_{n,k}) \qquad \text{(quasi-linear input)},$$

$$u_n = f_n(v_n) \quad \text{where} \quad v_n \equiv w_{n,0} + \sum_{(k,n)\in\mathcal{A}} u_k w_{n,k} \qquad \text{(linear input)}.$$

The linear input form corresponds to the standard linear or sigmoid activation functions and the quasi-linear input form corresponds to radial basis nodes which in general take a weighted sum of squared differences between inputs and some "mean". One exception to these forms is Softmax [Bri89]. The normalizing effect of Softmax nodes is readily absorbed into the energy or cost function for training, so these nodes are not necessary in the network.

Second derivatives are to be calculated of the cost function or energy function summed across all training patterns. Some standard cost functions are mean-square error or cross-entropy. So we are interested in the sum of terms for each training pattern of the form

$$\frac{\partial^2 E}{\partial w_{m,i} \partial w_{n,j}}$$

where E is the cost function or energy function for a single pattern, which itself is a function of the network outputs for the pattern.

2.3 Exact calculations

This section presents an exact algorithm for calculating second derivatives of the energy or cost for a single pattern. Basic derivatives can be calculated as follows. This assumes nothing about the form of the activation functions (and follows directly from the chain rule).

Lemma 2.3.1 *Assume a neural network framework as set up previously with the network a directed acyclic graph. Consider second derivatives of E which can be any function of activations from the output nodes. Equation (2.1) assumes that node n is not an output node. Equation (2.2) assumes there is no sequence of connections from m to n (i.e. u_n is not a partial function of u_m).*

$$\frac{\partial^2 E}{\partial u_m \partial u_n} = \sum_{(n,l)\in\mathcal{A}} \frac{\partial^2 E}{\partial u_m \partial u_l}\frac{\partial u_l}{\partial u_n} + \sum_{(n,l)\in\mathcal{A}} \frac{\partial E}{\partial u_l}\frac{\partial^2 u_l}{\partial u_m \partial u_n}\,, \tag{2.1}$$

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{m,i}\partial w_{n,j}} &= \frac{\partial^2 E}{\partial u_m \partial u_n}\frac{\partial u_m}{\partial w_{m,i}}\frac{\partial u_n}{\partial w_{n,j}} + 1_{m=n}\frac{\partial E}{\partial u_m}\frac{\partial^2 u_m}{\partial w_{m,i}\partial w_{m,j}} \\ &\quad + \frac{\partial E}{\partial u_m}\frac{\partial u_n}{\partial w_{n,j}} \sum_{(l,m)\in\mathcal{A}} \frac{\partial^2 u_m}{\partial w_{m,i}\partial u_l}\frac{\partial u_l}{\partial u_n}\,,\end{aligned} \tag{2.2}$$

where the indicator function $1_{m=n}$ takes the value 1 if $m = n$ and 0 otherwise.

Notice the final summation is only present if there is a sequence of connections from node n to node m.

Corollary 2.3.1 *Assume that for every node activation input is quasi-linear. Consider how the above second derivatives evaluate. Equation (2.3) assumes there is no sequence of connections from m to n.*

$$\begin{aligned}\frac{\partial^2 E}{\partial w_{m,i}\partial w_{n,j}} &= R_{m,n}\frac{\partial u_m}{\partial w_{m,i}}\frac{\partial u_n}{\partial w_{n,j}} + 1_{i\neq 0}\frac{\partial E}{\partial u_m}\frac{\partial u_m}{\partial v_m}\frac{\partial^2 v_m}{\partial u_i \partial w_{m,i}}\frac{\partial u_i}{\partial u_n}\frac{\partial u_n}{\partial w_{n,j}} \\ &\quad + 1_{m=n}1_{ij(i-j)=0}\frac{\partial E}{\partial u_m}\left(\frac{\partial^2 u_m}{\partial w_{m,i}\partial w_{m,j}}\Big/\frac{\partial u_m}{\partial w_{m,i}}\Big/\frac{\partial u_m}{\partial w_{m,j}} - \frac{\partial^2 u_m}{\partial v_m^2}\Big/\left(\frac{\partial u_m}{\partial v_m}\right)^2\right)\frac{\partial u_m}{\partial w_{m,i}}\frac{\partial u_n}{\partial w_{n,j}} \\ &\quad + 1_{m\neq n}1_{i=0}\frac{\partial E}{\partial u_m}\left(\frac{\partial^2 u_m}{\partial v_m \partial w_{m,i}}\Big/\frac{\partial u_m}{\partial v_m}\Big/\frac{\partial u_m}{\partial w_{m,i}} - \frac{\partial^2 u_m}{\partial v_m^2}\Big/\left(\frac{\partial u_m}{\partial v_m}\right)^2\right)\frac{\partial u_m}{\partial u_n}\frac{\partial u_m}{\partial w_{m,i}}\frac{\partial u_n}{\partial w_{n,j}}\end{aligned} \tag{2.3}$$

where $1_{ij(i-j)=0}$ denotes i or j is zero or $i = j$, and

$$R_{m,n} = \sum_{(n,l),(m,l)\in\mathcal{A}^*} G_l\frac{\partial u_l}{\partial u_m}\frac{\partial u_l}{\partial u_n} + \sum_{l,k\in output\text{-}nodes} \frac{\partial^2 E}{\partial u_l \partial u_k}\frac{\partial u_l}{\partial u_m}\frac{\partial u_k}{\partial u_n}\,, \tag{2.4}$$

$$G_m = \frac{\partial E}{\partial u_m}\frac{\partial^2 u_m}{\partial^2 v_m}\Big/\left(\frac{\partial u_m}{\partial v_m}\right)^2 + \sum_{(m,l)\in\mathcal{A}} \frac{\partial E}{\partial u_l}\frac{\partial u_l}{\partial v_l}\frac{\partial^2 v_l}{\partial u_m^2}\,, \tag{2.5}$$

and when n is not an output node

$$R_{m,n} = G_m\frac{\partial u_m}{\partial u_n} + \sum_{(n,l)\in\mathcal{A}} R_{m,l}\frac{\partial u_l}{\partial u_n}\,. \tag{2.6}$$

Proof First, when there is no sequence of connections from m to n, from Equation (2.1) we can prove by induction that

$$\begin{aligned}\frac{\partial^2 E}{\partial u_m \partial u_n} &= \frac{\partial u_m}{\partial u_n}\sum_{(m,l)\in\mathcal{A}} \frac{\partial E}{\partial u_l}\frac{\partial u_l}{\partial v_l}\frac{\partial^2 v_l}{\partial u_m^2} + \sum_{(n,l),(m,l)\in\mathcal{A}^+} G_l\frac{\partial u_l}{\partial u_m}\frac{\partial u_l}{\partial u_n} \\ &\quad + \sum_{l,k\in output\text{-}nodes} \frac{\partial^2 E}{\partial u_l \partial u_k}\frac{\partial u_l}{\partial u_m}\frac{\partial u_k}{\partial u_n}\,.\end{aligned}$$

If we now define

$$R_{m,n} \equiv \frac{\partial^2 E}{\partial u_m \partial u_n} + \frac{\partial E}{\partial u_m}\frac{\partial^2 u_m}{\partial v_m^2} \Bigg/ \left(\frac{\partial u_m}{\partial v_m}\right)^2 \frac{\partial u_m}{\partial u_n} , \tag{2.7}$$

we have proven Equation (2.6) for when there is no sequence of connections from m to n. Substituting in the definition of $R_{m,n}$ into Equation (2.3), and cancelling out terms almost gives us Equation (2.2). Equivalence of Equations (2.3) and (2.2) is shown by first substituting into Equation (2.2)

$$\begin{aligned}\sum_{(l,m)\in\mathcal{A}} \frac{\partial^2 u_m}{\partial w_{m,i}\partial u_l}\frac{\partial u_l}{\partial u_n} &= \sum_{(l,m)\in\mathcal{A}} \frac{\partial}{\partial w_{m,i}}\left(\frac{\partial u_m}{\partial v_m}\frac{\partial v_m}{\partial u_l}\right)\frac{\partial u_l}{\partial u_n} \\ &= \frac{\partial^2 u_m}{\partial w_{m,i}\partial v_m}\frac{\partial v_m}{\partial u_n} + \frac{\partial u_m}{\partial v_m}\frac{\partial^2 v_m}{\partial w_{m,i}\partial u_i}\frac{\partial u_i}{\partial u_n} .\end{aligned}$$

Substituting in the value of $R_{m,n}$ into Equation (2.3), noting that the indicators $1_{ij(i-j)=0}$ and $1_{i=0}$ are unnecessary in the last two terms, and rearranging proves equivalence. Also, we also get Equation (2.4) by induction from Equation (2.6). Notice Equation (2.4) is symmetric in n and m so we can drop the conditions on n and m for this equation. A final induction proof then shows Equation (2.6) for when there is a sequence of connections from m to n. □

Notice that if nodes use activation functions with linear input such as a sigmoid, then the corollary simplifies further because $\frac{\partial^2 v_l}{\partial u_m^2} = 0$, $\frac{\partial^2 v_l}{\partial u_m \partial w_{l,m}} = 1$ if $(m,l) \in \mathcal{A}$, etc.

Corollary 2.3.2 *Assume that for every node activation input is linear. Then the formula in Corollary 2.3.1 evaluate to, where Equation (2.8) assumes there is no sequence of connections from m to n:*

$$\begin{aligned} G_m &= \frac{\partial E}{\partial u_m}\frac{\partial^2 u_m}{\partial v_m^2} \Bigg/ \left(\frac{\partial u_m}{\partial v_m}\right)^2 , \\ \frac{\partial^2 E}{\partial w_{m,i}\partial w_{n,j}} &= R_{m,n}\frac{\partial u_m}{\partial w_{m,i}}\frac{\partial u_n}{\partial w_{n,j}} + 1_{i\neq 0}\frac{\partial E}{\partial u_m}\frac{\partial u_m}{\partial v_m}\frac{\partial u_i}{\partial u_n}\frac{\partial u_n}{\partial w_{n,j}} . \end{aligned} \tag{2.8}$$

For most energy functions, the second summation in Equation (2.4) is easy to compute since usually $\frac{\partial^2 E}{\partial o_i \partial o_j} = 0$ for $i \neq j$. For instance, for mean-square error on multiple outputs, we get

$$\frac{1}{\sigma^2}\sum_{l\in input\text{-}nodes} \frac{\partial u_l}{\partial w_{m,i}}\frac{\partial u_l}{\partial w_{n,j}} + \text{constant} .$$

To calculate second derivatives using any of the above equations, all first derivatives between nodes have to be computed, that is, we need $\frac{\partial u_l}{\partial u_m}$ for all $(m,l) \in \mathcal{A}^*$. This applies even if we are only interested in the diagonal terms of the second derivative. Notice that for networks with two or less hidden layers, all these first derivatives will be available once all first derivatives of output nodes are computed. First derivatives $\frac{\partial u_l}{\partial u_m}$ for a fixed l can be calculated by backward propagation of derivatives from l. Derivatives $\frac{\partial u_l}{\partial u_m}$ for a fixed m can be calculated by forward propagation of derivatives from l.

Also, notice that if node m is an output node, then l in the summation in Equation (2.4) can only be m so $R_{m,n}$ can be computed in constant time, assuming the required first derivatives exist. Second derivatives are therefore only expensive to calculate exactly if both weights are in hidden nodes. Similarly, second derivatives are quite efficient to calculate exactly if one of the weights is at most one or two node away from an output node.

We therefore get the following algorithm for exact computation of second derivatives:

1. Calculate G_m for each node m. (This takes in the order of one back-propagation cycle to do.)

2. For each node n calculate $R_{m,n}$ for relevant nodes m:

   (a) Forward propagate from n to calculate derivatives $\frac{\partial u_m}{\partial u_n}$ for each node m using the standard formula
   $$\frac{\partial u_m}{\partial u_n} = \sum_{(m,l)\in\mathcal{A}} \frac{\partial u_l}{\partial u_n}\frac{\partial u_m}{\partial u_l}$$

   (b) If we are only interested in $R_{m,n}$ for $n = m$, then compute $R_{m,m}$ using Equation (2.4).

   (c) Else, backward propagate starting from the output nodes to calculate $R_{m,n}$ for each node m using Equation (2.6) for the recursive case and Equation (2.4) for the base case.

This requires at the worst case approximately $2K$ back/forward-propagation cycles. Any required second derivative can now be read from the stored calculations in constant time.

Notice that for the special case where we are only after the block diagonals $\frac{\partial^2 E}{\partial w_{m,i}\partial w_{n,j}}$ for $n = m$, of a network with at most two hidden layers, then calculation of $R_{n,n}$ for every node n takes only (approximately) three back-propagation cycles since first derivatives can be calculated with one back-propagation cycle, and each $R_{n,n}$ can be calculated directly from Equation (2.4).

2.4 Approximations

If we assume nodes m and n are at the same level, then from Equation (2.1) we get

$$\frac{\partial^2 E}{\partial u_m \partial u_n} = \sum_{(m,l),(n,k)\in\mathcal{A}} \frac{\partial^2 E}{\partial u_l \partial u_k}\frac{\partial u_l}{\partial u_m}\frac{\partial u_k}{\partial u_n} + \sum_{(m,l),(n,l)\in\mathcal{A}} \frac{\partial E}{\partial u_l}\frac{\partial^2 u_l}{\partial u_m \partial u_n} .$$

Le Cun *et al.*'s approximation corresponds to the case where we have linear input nodes and we assume $\frac{\partial^2 E}{\partial u_l \partial u_k} = R_{l,k} = 0$ for nodes $l \neq k$ with l and k at the same level, giving the rule (they use a different parameterization so their form is different):

$$\frac{\partial^2 E}{\partial u_m^2} = \sum_{(m,l)\in\mathcal{A}} \left(\frac{\partial^2 E}{\partial u_l^2}\left(\frac{\partial u_l}{\partial v_l}\right)^2 w_{l,m}^2 + \frac{\partial E}{\partial u_l}\frac{\partial^2 u_l}{\partial v_l^2} w_{l,m}^2 \right) ,$$

$$\frac{\partial^2 E}{\partial w_{m,i}^2} = \left(\frac{\partial^2 E}{\partial u_m^2}\left(\frac{\partial u_m}{\partial v_m}\right)^2 + \frac{\partial E}{\partial u_m}\frac{\partial^2 u_m}{\partial v_m^2} \right) u_i^2 .$$

These can be computed with one back-propagation cycle so calculation is efficient [LDS90]. MacKay, however, found the method inaccurate for his purposes [Mac91] and instead dropped the first summation from Equation (2.4) and all but the first term from Equation (2.3) leaving a calculation that requires only the first derivatives,

$$\frac{\partial^2 E}{\partial w_{m,i}\partial w_{n,j}} \approx \sum_{l,k \in \textit{output-nodes}} \frac{\partial^2 E}{\partial u_l \partial u_k}\frac{\partial u_l}{\partial w_{m,i}}\frac{\partial u_k}{\partial w_{n,j}} .$$

He reports this is inaccurate when approximating the determinant of the matrix of second derivatives, or looking at individual second derivatives, but seems fine when approximating the trace of the matrix. This approximation, ignoring the second derivatives, corresponds to the Levenberg-Marquardt approximation in non-linear least squares [PFTV88, p523]. Clearly, better approximations involving a few more terms are available.

A second form of approximation exists if the network cost function being minimized corresponds to the negative logarithm of the likelihood of the training sample, as is often the case when using mean-square error or cross-entropy cost functions [BW91a, BW87, EJM90]. Suppose the likelihood of the training sample of size D is given as a product over patterns (x_d, y_d) in the sample

$$p(\vec{y} \mid \vec{x}, w) = \prod_{d=1,\ldots,D} l(y_d \mid x_d, w) ,$$

where $l(y_i \mid x_i, w)$ is the likelihood of the d-th pattern and is a function of the network output for input x_i and the "correct" output y_i. Then maximum likelihood training (maximum *a posterior* training is similar) corresponds to minimizing the energy function with component for each pattern

$$E_d = -\log l(y_d \mid x_d, w) .$$

For instance, when using the mean-square error cost function on a single output, we are implicitly using the Gaussian likelihood

$$l(y_d \mid x_d, w) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_d - o_d)^2\right) ,$$

where σ is the standard deviation and o_d is the network output for input x_d.

When network weights are near a local maxima of the likelihood the second derivatives can be approximated using the following formula:

$$\begin{aligned}
\sum_{d=1,\ldots,D} \frac{\partial^2 E_d}{\partial w_{m,i}\partial w_{m,j}} &= \frac{\partial^2 - \log p(\vec{y} \mid \vec{x}, w)}{\partial w_{m,i}\partial w_{m,j}} \\
&= \sum_{d=1,\ldots,D} \frac{\partial^2 - \log l(y_d \mid x_d, w)}{\partial w_{m,i}\partial w_{m,j}} \\
&\approx \sum_{d=1,\ldots,D} \frac{\partial \log l(y_d \mid x_d, w)}{\partial w_{m,i}}\frac{\partial \log l(y_d \mid x_d, w)}{\partial w_{m,j}} \\
&= \sum_{d=1,\ldots,D} \frac{\partial E_d}{\partial w_{m,i}}\frac{\partial E_d}{\partial w_{m,j}} .
\end{aligned} \tag{2.9}$$

Notice this only requires calculation of the first derivatives of the energy. The approximation step is justified because at a local maxima, on the assumption that the weights w are

approximately correct (which they may well not be) and the sample size D is large [Ame85, p.14], the two sides are approximately equal. This is the approximation used in the "scoring" method of maximum likelihood training [Ame85]. This approximation is best used during search when fast estimates of second derivates are required. The approximation would be misleading when after good approximations for error bars because it assumes that the thing we are attempting to evaluate is approximately true (that the weights are correct).

A third more exact approximation of second derivatives can be done by numerical differentiation of the first derivatives, which in turn can be calculated using standard back-propagation. This corresponds to

$$\frac{\partial^2 E(w)}{\partial w_{m,i} \partial w_{n,j}} \approx \frac{1}{\Delta w_{m,i}} \left(\frac{\partial E(w + \Delta w_{m,i})}{\partial w_{n,j}} - \frac{\partial E(w)}{\partial w_{n,j}} \right) .$$

If there are $|w|$ different weights in the network, then this requires $|w| + 1$ back-propagation cycles to compute the necessary first derivatives (compared to $2K$ back/forward-propagation cycles for the exact calculation, where K is the number of nodes). This method has been used in the case where there is a small number of weights by MacKay [Mac91].

A more efficient approach is to use numerical differentiation to calculate second derivatives of the activations, $\frac{\partial^2 E}{\partial u_m \partial u_n}$, in $K+1$ back-propagation cycles, and to calculate additional first derivatives $\frac{\partial u_m}{\partial w_n}$ as required in approximately $\frac{K}{2}$ back-propagation cycles, and then use Equations (2.3) and (2.7). A similar method is suggested in [BL88]. This approximation is therefore of the same computational order as the exact calculations described previously, but requires little additional algorithm overhead other than the back-propagation algorithm.

Acknowledgements

Thanks to Sue Becker, Yann Le Cunn and David MacKay for useful feedback.

Bibliography

[Ame85] T. Amemiya. *Advanced Econometrics.* Harvard University Press, Cambridge, MA, 1985.

[BB88] A.R. Barron and R.G. Barron. Statistical learning networks: A unifying view. In *1988 Symposium on the Interface: Statistics and Computing Science*, Reston, Viginia, 1988.

[BC91] A.R. Barron and T.M. Cover. Minimum complexity density estimation. *IEEE Trans. on IT*, ??, 1991.

[Ber85] J. O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, New York, 1985.

[BH89] Eric B. Baum and David Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[BL88] S. Becker and Y. Le Cun. Improving the convergence of back-propagation learning with second order methods. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 29–37. Morgan Kaufmann, 1988.

[Bri89] J.S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fougelman-Soulie and J. Hérault, editors, *Neuro-computing: Algorithms, Architectures and Applications.* Springer-Verlag, 1989.

[Bun90] W.L. Buntine. Learning classification trees. Technical Report FIA-90-12-19-01, RIACS and NASA Ames Research Center, Moffett Field, CA, 1990. Paper presented at Third International Workshop on Artificial Intelligence and Statistics.

[Bun91] W.L. Buntine. *A Theory of Learning Classification Rules.* PhD thesis, University of Technology, Sydney, 1991.

[BW87] E.B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D.Z. Anderson, editor, *Neural Information Processing Systems (NIPS)*, pages 52–61, 1987.

[BW91a] W.L. Buntine and A.S. Weigend. Bayesian back-propagation. Submitted, 1991.

[BW91b] W.L. Buntine and A.S. Weigend. Calculating second derivatives on feed-forward networks. Submitted, 1991.

[Cot90] N.E. Cotter. The Stone-Weierstrass theorem and its application to neural networks. *IEEE Trans. on Neural Networks*, 1(4):290–295, 1990.

[DL91] John S. Denker and Yann Le Cun. Transforming neural-net output levels to probability distributions. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, page 853. Morgan Kaufmann, 1991.

[EJM90] A. El-Jaroudi and J. Makhoul. A new error criterion for posterior probability estimation with neural nets. In *Int. Joint Conf. on Neural Networks*, pages III–185–192, San Diego, CA, 1990.

[Gol88] R.M. Golden. A unified framework for connectionist systems. *Biological Cybernetics*, 59:109–120, 1988.

[Goo83] I.J. Good. *The Goodbook*. University of Minnesota Press, Minnesota, 1983.

[Hau91] D. Haussler. A decision theoretic generalization of the PAC learning model and its application to some feed-forward neural networks. *Information and Control*, 1991. To appear.

[HHL86] E.J. Horvitz, D.E. Heckerman, and C.P. Langlotz. A framework for comparing alternative formalisms for plausible reasoning. In *Fifth National Conference on Artificial Intelligence*, pages 210–214, Philadelphia, 1986.

[HP90] J.B. Hampshire II and B.A. Pearlmutter. Equivalence proofs for multi-layer perceptron classifiers and the Bayesian discrimination function. In David S. Touretzky, Jeffrey L. Elman, Terrence J. Sejnowski, and Geoffrey E. Hinton, editors, *Proceedings of the 1990 Connectionist Models Summer School*. Morgan Kaufmann, 1990.

[HT90] T.J. Hastie and R.J. Tibshirani. *Generalised Additive Models*. Chapman and Hall, London, 1990.

[Ish90] M. Ishikawa. A structural learning algorithm with forgetting of link weights. Technical Report TR-90-7, Electrotechnical Laboratory, Life Electronics Research Center, Tokyo, 1990. Modified version of a paper presented at *IJCNN*, Washington D.C. 1989.

[JJNH91] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1), 1991.

[LDS90] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (NIPS*89)*, page 589. Morgan Kaufmann, 1990.

[LeC88] Y. Le Cun. A theoretical framework for back-propagation. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 21–28. Morgan Kaufmann, 1988.

[LeC89] Y. Le Cun. Generalization and network design strategies. Technical Report CRG-TR-89-4, Dept. of Computer Science, University of Toronto, Toronto, M5S 1A4, Canada, 1989.

[LTS89] E. Levi, N. Tishby, and S.A. Solla. A statistical approach to learning and generalization in layered neural networks. In R. Rivest, D. Haussler, and M.K. Warmuth, editors, *COLT'89: Second Workshop on Computational Learning Theory*, pages 245–260, University of California, Santa Cruz, 1989. Morgan Kaufmann.

[Mac91] D.J.C. Mackay. A practical Bayesian framework for backprop networks. Submitted to *Neural Computation*, 1991.

[MN89] P. McCullagh and J.A. Nelder. *Generalised Linear Models*. Chapman and Hall, London, second edition, 1989.

[OH91] M. Opper and D. Haussler. Generalised performance of Bayes optimal classification algorithm for learning a perceptron. In *COLT'91: 1991 Workshop on Computational Learning Theory*. Morgan Kaufmann, 1991. Manuscript.

[Pan89] Panel. Discriminant analysis and clustering. *Statistical Science*, 4(1):34–69, 1989.

[Pea88] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan and Kauffman, 1988.

[PFTV88] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 1988.

[Pre89] S.J. Press. *Bayesian Statistics*. Wiley, New York, 1989.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui89] J.R. Quinlan. Unknown attribute values in induction. In *Proceedings of the Sixth International Machine Learning Workshop*, Cornell, New York, 1989. Morgan Kaufmann.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, page 318. MIT Press, 1986.

[Ris87] J. Rissanen. Stochastic complexity. *J. Roy. Statist. Soc. B*, 49(3):223–239, 1987.

[SST91] H.S. Seung, H. Sompolinsky, and N. Tishby. Statistical mechanics of learning from examples; i. general formulation and annealing approximation. Manuscript, 1991.

[TSN90] G.G. Towell, J.W. Shavlik, and M.O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Eighth National Conference on Artificial Intelligence*, pages 861–866, Boston, Massachusetts, 1990.

[Vap82] V. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, New York, 1982.

[WF87] C.S. Wallace and P.R. Freeman. Estimation and inference by compact encoding. *J. Roy. Statist. Soc. B*, 49(3):240–265, 1987.

[WHR90] Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart. Predicting the future: a connectionist approach. *International Journal of Neural Systems*, 1:193–209, 1990.

[WRH91] Andreas S. Weigend, David E. Rumelhart, and Bernardo A. Huberman. Generalization by weight-elimination with application to forecasting. In Richard P. Lippmann, John Moody, and David S. Touretzky, editors, *Advances in Neural Information Processing Systems 3 (NIPS*90)*. Morgan Kaufmann, 1991.

[Zel90] A. Zellner. Bayesian methods and entropy in economics and econometrics. In W.T. Grandy, Jr. and L. Schlick, editors, *Maximum Entropy and Bayesian Methods*. Kluwer, 1990.